AN ENSEMBLE MACHINE VISION SYSTEM FOR AUTOMATED DETECTION OF SURFACE DEFECTS IN AIRCRAFT PROPELLER BLADES

A Dissertation
Presented to
The Academic Faculty

by

Ivan Ren

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
George W. Woodruff School of Mechanical Engineering

Georgia Institute of Technology
May, 2020

AN ENSEMBLE MACHINE VISION SYSTEM FOR AUTOMATED DETECTION OF SURFACE DEFECTS IN AIRCRAFT PROPELLER BLADES

Approved by:

Dr. Thomas Kurfess, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Christopher Saldana, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Katherine Fu
School of Mechanical Engineering
Georgia Institute of Technology

Date Approved: April 20, 2020

ACKNOWLEDGEMENTS

First, I would like to thank Dr. Saldana, for his continuous support and guidance. I've learned a lot through our interactions and am extremely grateful for the opportunities to explore new research areas and gain invaluable experience. I would also like to thank Dr. Kurfess and Dr. Fu for their time, commitment, and expertise throughout the research.

Secondly, I would like to thank my partners at Robins Air Force base for their support and technical insights on this project. They have been crucial to the success of my research and I am grateful for the opportunity.

Finally, I would like to thank my fellow labmates for their help and support. They have been a constant source of laughter and entertainment, even in difficult times.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

CNN	Convolutional Neural Network
ML	Machine Learning
DL	Deep Learning
kNN	k-Nearest Neighbor
ANN	Artificial Neural Networks
SVM	Support Vector Machines
GLCM	Gray Level Co-occurrence Matrix
MLP	Multilayer Perceptron
GPU	Graphics Processing Unit
ReLU	Rectified Linear Unit
MSE	Mean Square Error
SGD	Stochastic Gradient Descent
ResNet	Residual Network

SUMMARY

Visual inspections comprise the majority of inspections for large transport aircraft and are traditionally conducted by human operators. The manual inspection process is time-consuming, inconsistent, and subject to human errors. Automated defect detection systems have been developed to leverage computer vision and deep learning to decrease inspection times and improve detection performance. Current state-of-the-art systems use convolutional neural networks (CNNs) to detect defects from image data. The performance of these systems is insufficient for critical aircraft inspection and there is little consideration for the balance of false alarms and missed detections. This thesis presents a novel application of deep learning ensembles to automated aircraft visual inspection to improve the performance of CNNs and provide a framework for managing the tradeoff between Type I and Type II error. Stacked ensembles are constructed from three single model CNNs and a logistic regression meta-learner is used to combine their predictions. The dataset is generated from images taken at a borescope inspection process for C130 propeller blades. Transfer learning and data augmentation are used to supplement the size and diversity of the dataset. The performance of the stacked ensembles is evaluated, and it is found that stacked ensembles of CNNs outperform the current state-of-the-art defect detection approaches. The stacked ensembles achieved accuracy, precision, and recall of 99.8, 99.61, and 100 percent respectively on the test images. Furthermore, it is shown that with sufficient error diversity, ensembling can be used to create systems that eliminate Type I or Type II errors in the testing set. The overall error reduction provided by ensembles allow for lower rates of false alarms at prediction thresholds that result in no missed detections.

CHAPTER 1. INTRODUCTION

Inspections play an important role in aircraft maintenance and are crucial in ensuring the airworthiness of a plane. One of the most common inspection techniques is visual inspection. Visual inspections comprise over 80% of large transport aircraft inspections and are often the fastest and most economical way of providing early detection of defects before they reach critical sizes [1]. Typical aircraft defects found through this process include cracks, corrosion, disbonding, missing or deformed parts, and incidental damage. Traditionally, visual inspections have been conducted by human operators with the assistance of visual aids such as mirrors, borescopes, and optical cameras that allow operators to inspect hard-to-access areas without the deconstruction of the component. This approach is mostly manual, making it time-consuming, tedious, and subjective. Additionally, operator performance can be negatively affected by a variety of personal and environmental factors with potentially catastrophic consequences [2]. In 2017, a fatal KC-130 crash caused by a corroded propeller blade was attributed to faulty inspection by civilian operators at Robins Air Force Base [3]. To overcome the limitations of human inspection, there have been multiple efforts to automate the inspection process using computer vision. Applications within aircraft maintenance have been limited and the majority of these techniques have been developed for analogous domains that suffer from similar defects and also rely heavily on human inspection for maintenance. These domains include naval inspection, structural health monitoring, and steel manufacturing. The automated methods previously developed can be loosely categorized into texture analysis, color analysis, machine learning (ML), and deep learning (DL) methods. Texture and color

analysis methods, which include pixel level thresholding, morphological operations, and wavelet transforms, seek to identify and extract local irregularities from images [4]. These methods are difficult to apply to a wide range of defects and often require precise calibration and setup. ML methods aim to create high performing predictive models from relevant features extracted from data. One challenge in applying ML methods is the reliance on domain experts with intimate knowledge of the application to perform feature extraction. DL, a subset of ML, circumvents the need for domain experts and is capable of learning high level features independently. It has proven its capabilities in a variety of imaging related tasks but is rarely applied to automated defect detection due to a lack of sufficient datasets [5]. The generalization performance of different DL models can also vary depending on the construction of their training sets and the parameters defined during the training process. Generalization refers to a model's ability to make accurate predictions on new data drawn from the same distribution as the training data. Furthermore, choosing the proper size and structure of a DL model is often a time-consuming process and the selected model may not be as effective on operational data as it was on training data [5].

1.1 Problem Statement

The objective of this study is to improve the generalization performance of deep learning models applied to aircraft defect detection by combining the outputs of multiple models through ensembling techniques. An understanding of how ensemble methods impact detection performance in visual aircraft inspection has not been well addressed in the literature. The present thesis seeks to answer the research question of whether ensemble methods can be adequately configured to improve the detection performance of CNNs for image defect detection algorithms in visual aircraft inspection. To address this question,

multiple state-of-the-art CNNs were trained and evaluated on images containing typical aircraft defects found in C130 propeller blades from a borescope inspection process at Robins Air Force Base. A second level ML algorithm was trained to combine the outputs of various CNNs, resulting in an increase in generalization performance. Furthermore, like human inspection, automated detection methods cannot claim to be error-free. There is a tradeoff between Type I and Type II error that must be managed to maintain confidence in the algorithm and prevent critical missed detections. The ensembling methodology proposed can be used to construct DL architectures that minimize Type I or Type II error depending on the application and criticality of the inspection process.

1.2 Structure

The remainder of this thesis is organized as follows. Chapter 2 details the visual inspection process and highlights the performance factors that can be alleviated by the development of an automated inspection framework. This is followed by a literature review of the current state of the art in automated defect detection and an introduction of the technologies and terminologies used in this study. Chapter 3 describes the proposed ensembling technique and the process of selecting, training, and evaluating different ensemble components, including the base CNNs and the second level learner. Chapter 4 presents the results of the various models on test cases and discusses the performance differences between single model CNNs and ensembles. Finally, in Chapter 5, the contributions of this study are discussed, and recommendations are made for future research.

CHAPTER 2. BACKGROUND

The importance of visual inspection in the aircraft maintenance system has driven multiple efforts to develop computer vision-based methodologies to improve inspection performance and reliability. This chapter reviews the current state of visual inspection and past frameworks developed for automated defect detection. The chapter concludes with a brief introduction of the technologies and terminologies used in this study.

2.1 Visual Inspection

Visual inspection is ubiquitous within aviation maintenance and is valued for its flexibility in the number and types of defect indications it can detect [6]. The cornerstone of the process is the human operator. Recent innovations have automated the scanning of aircraft surfaces through imaging technology but there is no widespread substitute for the ability of a trained inspector to detect irregularities from the images or video taken [7]. The reliance on the decision-making ability of humans in visual inspection has encouraged multiple studies into the personal and environmental factors that affect inspector performance.

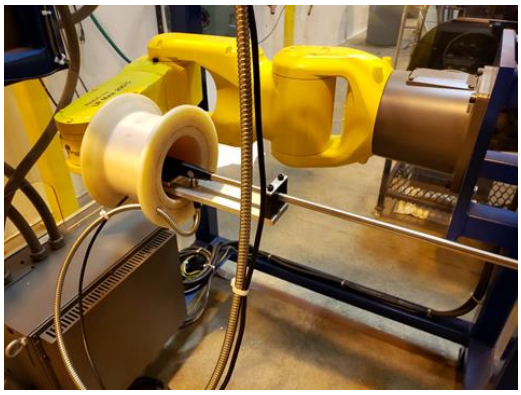

Figure 2.1: Robotic arm and borescope used in visual inspection

2.2 Performance Factors in Visual Inspection

2.2.1 Defect Rate

Defect rates are defined as the probability of a defect occurring within the section being inspected. Studies have shown that inspection accuracy decreases with the defect rate and that the rate of missed detections and false alarms increase as defect rates decrease [8]. In many inspections, defects exist in only a small subset of the total inspected area and operators must often examine hundreds or thousands of images before a defect is identified. Automated detection systems provide consistent performance irrespective of defect rates; however, low defect rates increase the time and effort needed to obtain sufficient samples to develop accurate algorithms.

2.2.2 Fatigue

Mental fatigue is frequently a factor in the visual inspection process as operators are tasked with remaining alert and attentive to any defects for the duration of their shifts. It can result from a combination of lack of sleep, job pressure, low defect rates, prolonged time on task, high memory loads, presence of ambiguous irregularities, or the absence of feedback [8, 9]. In most cases, the deterioration in vigilance is complete within the first 30 minutes of an inspection task and tend to be worse for difficult detection tasks [10]. Operators suffering from fatigue experience a decrease in performance over time and are likely to report fewer detections and false alarms. Some strategies to combat mental fatigue listed in [8] include limiting the inspection period to 30 minutes and rotating tasks. These strategies require frequent stops and productivity is lost as operators reacclimate to the task after their breaks.

2.2.3 Feedback

Feedback is an essential component in any training task and has shown positive results in human inspection when it is provided in a timely and appropriate manner [8]. In [11], subjects were trained in an aircraft inspection task and it was found those that were provided feedback demonstrated substantial improvements in inspection performance over the control group in terms of speed and accuracy. In aircraft maintenance, feedback is rare, and operators are not provided the information needed to adjust inspection approaches. If a defect is missed, it is often only detected in a separate inspection down the line, or sometimes, not at all. An automated defect detection system implemented in parallel with human inspectors can provide immediate feedback on the performance of the operator and serve as a tool in training new personnel.

2.2.4 Computer Vision Methodologies for Defect Detection

There have been significant advances in surface inspection using computer vision in recent years. These techniques have the potential to perform autonomous defect detection at significantly decreased process times while maintaining consistent performance, irrespective of the various factors that hinder human inspection. Few of these techniques have been directly applied to aircraft maintenance but the typical defects found on aircraft are congruous with those found in other fields such as steel manufacturing and structural health monitoring. The related works in this study focus on computer vision methods that detect defects from images as imaging technology is becoming increasingly popular in inspection due to its speed and safety. These methods include both binary detections, where the goal is to separate images into defect and defect-free categories, and multi-class detection, where defects are further classified by their type and description. The most common methods in the literature are described below.

Table 1: List of defect detection methods reviewed

Approach	Method	References
Textural & Color Analysis	1. Histogram properties	[12, 13]
	2. Co-occurrence matrix	[14, 15, 16, 17]
	3. Structural methods	[18, 19, 20]
	4. Spatial/Frequency domain analysis	[21, 22, 23, 24]
Machine Learning	1. k-Nearest Neighbors (kNN)	[25, 26]
	2. Artificial Neural Networks (ANN)	[27]
	3. Support Vector Machines (SVM)	[28, 29, 30]
Deep Learning	1. CNNs	[32, 33, 34, 35]

2.2.5 Texture and Color Analysis

2.2.5.1 Histogram properties

Despite their simplicity, histogram techniques have proven their value in various applications, either in extracting low-level features or as a pre-processing technique. Histogram techniques are fast and easy to implement, making them suitable for real-time operations. They are also insensitive to geometric transformations and the spatial relationships between color pixels [4]. Commonly used histogram statistics include mean, median, standard deviation, and variance. Histograms techniques have been applied for both texture and color analysis. Lee et al. [12] created a corrosion detection algorithm based on a multivariate discriminant function using the histogram means and ranges of the red, green, and blue channels of an image. Ng [13] proposed a novel valley-emphasis thresholding technique to segment surface defects from their surroundings; however, it requires that the intensity of the defect is separable from that of the surrounding.

2.2.5.2 Co-occurrence matrix

Gray level co-occurrence matrices (GLCM), first defined in [14], are 2D matrices that are widely used in texture analysis. They measure the spatial dependency of two graylevels and are constructed by calculating how often a pixel of intensity *i* occurs in a specified spatial relationship to a pixel with intensity *j*. From the GLCM, features such as entropy, contrast, energy, and homogeneity can be extracted to perform texture analysis.

There have been several works using GLCMs to detect defects. Caleb and Steuer [15] extracted features from GLCMs calculated at various angles to train a multilayer perceptron (MLP) to detect defects in rolled steel. Tang et al. [16] classified defects using a NN from feature vectors extracted through edge detection and GLCMs to limited effect. Bonnin-Pascual and Ortiz [17] combined GLCM's and histogram thresholding within the HSV color space to detect corrosion and cracks in naval vessels. While popular, GLCMs suffer from a number of shortcomings. There is no generally accepted method for defining the spatial relationships used to generate the matrices and the number of graylevels are reduced to keep the sizes of the matrices manageable [4].

2.2.5.3 Structural Methods

Structural methods extract low-level textural elements and combine them with histogram properties to create models or general heuristics that define defects. These low-level elements include graylevel subregions or line segments defined by Hough transforms and edge filters. Wang and Cheng [18] used the circular Hough transform to detect pitting corrosion in microscopic images and used an equivalent circle technique to identify pits with irregular shapes. Guo et al. [19] proposed a hybrid image segmentation method based on edge detection and morphological erosion and dilation to detect scratches on steel surfaces that have low signal-to-noise ratios. Jeon et al. [20] defined a morphological

criterion of area ratio, compactness, roughness, and orientation to distinguish surface cracks from scales in steel billets. Structural methods suffer from similar shortcomings as other heuristic-based approaches in that their performance is dependent on the fidelity of their models. They are difficult to employ for multiple classes of defects as each defect would need to be defined by a unique set of heuristics.

2.2.5.4 Spatial and frequency domain analysis

A common characteristic of spatial and frequency domain analysis techniques is the application of filter banks to input images to measure the energy of the filter responses [4]. Filter banks are arrays of band-pass filters that separate the input into separate sub-frequency bands and can be used as a feature extractor or a signal denoiser. The energy of the filter responses is used to classify defects based on heuristic models or machine learning techniques. In [21], Mumtaz et al. proposed an automated inspection technique for aircraft skins using directional energies obtained from contourlet and discrete cosine transforms to differentiate between cracks and scratches. This method correctly identified 96.6% of crack images and 97.3% of scratch images. Liu et al. [22] presented a method to detect rivets in aircraft joints using Gabor filters and wavelet transforms that is insensitive to variations in illumination, surface reflectivity, and surface roughness unlike traditional methods such as Hough transforms and Canny edge detection. Their method was developed in support of a technique used to detect pillowing deformation caused by corrosion in the faying surfaces of riveted joints. Gunatilake et al. [23] used a discrete wavelet transform and a kNN classifier to detect corrosion on aircraft skins. A crack detection model using directional wavelet filters and a NN for classification was also described. Within the color space, Jahanshahi and Masri [24], evaluated the effects of different color spaces, number of features, and different sub-image windows on classification performance in wavelet analysis. It was found that systems that integrated CbCr color channels with textural features had improved performance on their detection task.

2.2.6 Machine Learning

Development of machine learning methods for automated defect detection has seen growth in recent years as the field has matured and computation resources have become cheap and abundant. Machine learning algorithms are prediction-oriented models that differ from the methods described in the previous sections as they do not rely on human-defined heuristics map inputs to outputs. Rather, they are trained on large quantities of historical data to learn and self-define an algorithm that can be used to make predictions on new data. As a result, most ML models are uninterpretable and unsuitable for understanding relationships. ML can be further categorized into supervised and unsupervised learning. In supervised learning, explicit labels are applied to inputs and outputs to develop a function to best approximate the relationship between the two. Unsupervised learning does not have labelled outputs and the algorithm is tasked with inferring the natural structure present in the data. ML algorithms are suitable for detecting defects in processes where the environment and defects may vary over time. Several ML techniques for automated defect detection are reviewed in the following section.

2.2.6.1 k-Nearest Neighbor (kNNs)

The k-nearest neighbor algorithm is a supervised classification algorithm that seeks to make predictions on a sample data point by determining the *k* nearest points by Euclidean distance. The most frequent class label among the *k* nearest neighbors is applied to the sample data point. In Figure 2.2, a kNN algorithm of *k* equal to 3 is shown. The datapoint denoted by the yellow triangle is assigned the class label of the green circles due to majority vote.

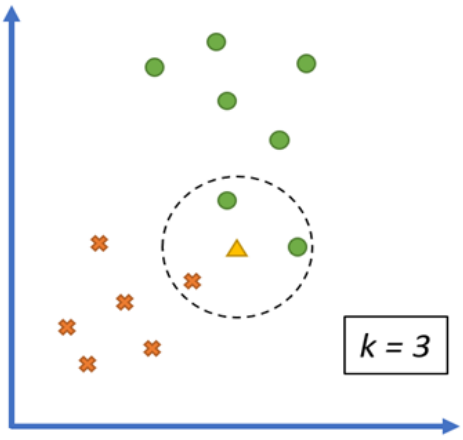

Figure 2.2: Sample kNN algorithm where *k* = 3

Mandriota et al. [25] compared the effect of features extracted by a Gabor filter and wavelet transform on the performance of kNN classifiers in detecting surface defects on railroad tracks. It was found that the Gabor filter maintained a low error rate as *k* approached infinity while for the wavelet transform, the error increased as *k* increased. In [26], Unsalan and Ercil applied a kNN classifier to features extracted from GLCMs, Markoff random fields, and histograms to detect rust on steel surfaces.

2.2.6.2 Artificial Neural Networks (ANNs)

Artificial neural networks are comprised of interconnected nodes or *neurons* with updateable weights organized in a series of layers. A typical ANN consists of an input layer, one or more hidden layers, and an output layer. As the network is trained, the weights of the neurons are typically modified via backpropagation to map inputs to outputs. Figure 2.3 shows an ANN structure with a single hidden layer.

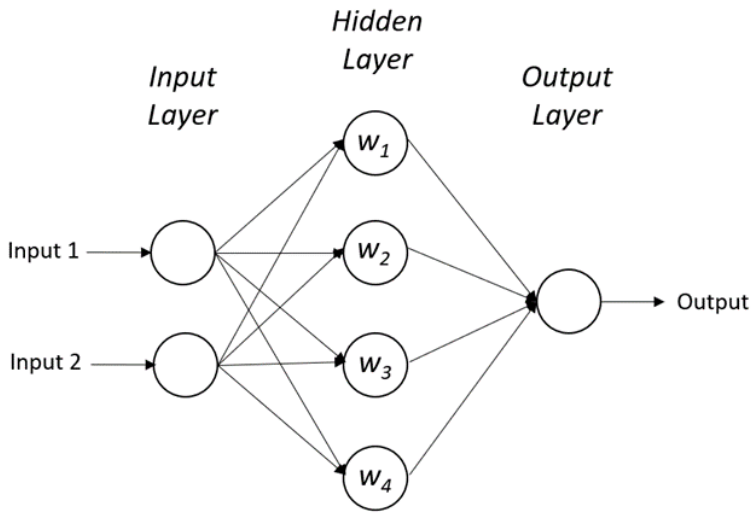

Figure 2.3: A simple ANN structure

Martins et al. [27] proposed a system for visual inspection of surface defects on rolled steel plates by passing images through a series of Gaussian filters, edge filters, and Hough transforms. Principal component analysis was conducted on the filtered images to select features for training an ANN. The proposed system achieved classification accuracies of around 98% on a test set of 300 images.

2.2.6.3 Support Vector Machines (SVMs)

A support vector machine is a supervised ML algorithm that seeks to determine the decision boundary or *hyperplane* that best separates a dataset into different classes and maximizes the margin between the hyperplane and nearest datapoints. SVMs are typically used in binary classification but can be expanded to multi-class classification through the use of an iterative one-versus-one approach or a one-versus-all classification criteria. Figure 2.4 illustrates the SVM algorithm for a linearly separable dataset. SVMs are unsuitable for large, noisy datasets and do not provide a probabilistic explanation for classification.

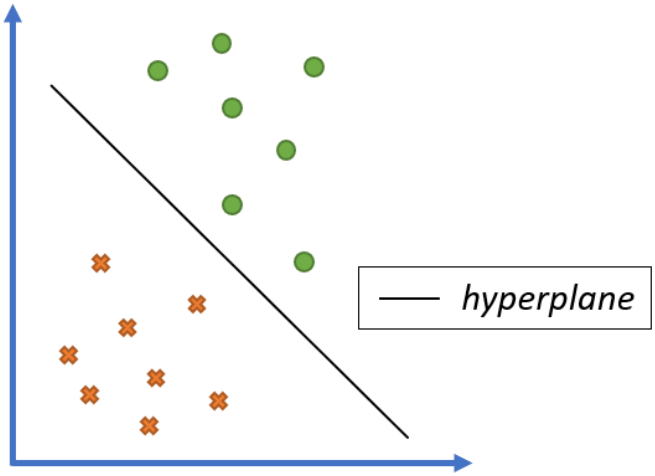

Figure 2.4: SVM on a linearly separable dataset

Jia et al. [28] trained an SVM to detect seams in rolled steel. An incremental learning algorithm was used to update the SVM as new defect images became available, allowing the algorithm to continue to learn over time. In [29], Hoang and Tran employed an SVM to differentiate between corroded and intact surfaces of sewer and water supply pipes. Histogram properties, GLCM-based features, and gray-level run lengths features formed the textural descriptors that were used to train the SVM and a sliding window approach was used to localize the corrosion within the input image. The proposed model achieved an accuracy level of 92.81% on the test data but had an accuracy level of 98.23% on the training data. This suggests that either the training data was a poor representation of the test data or the model was overfitted to the training data. Overfitting occurs when the generated model is too closely fit to the training data, making it a poor predictor for any new datapoints. Zhiwei et al. [30] developed a mobile robot platform for inspecting rivets on aircraft for cracks. An onboard camera unit is used to take pictures of the aircraft and a fuzzy-SVM algorithm is used to detect cracks and classify them based on the direction of crack growth. The one-versus-all approach was used to make the multi-class classification.

2.2.6.4 Deep Learning

Deep learning methods contain multiple layers of feature representation that amplify aspects of the input that are important for classification and suppress irrelevant variations such as differences in background, pose, lighting and surrounding objects [31]. These layers are not defined by domain experts and can instead be learned from data using a general learning procedure. DL is particularly applicable in fields, such as visual inspection, where humans are unable to explain their expertise. Through years of operating experience, seasoned inspectors have accumulated a trove of intangible knowledge and intuition that is unavailable to new trainees and cannot be captured in formalized training. DL has become increasingly popular in recent years due to advancements in graphics processing units (GPUs) that have made computation many times faster. A type of DL algorithm that has gained popularity in numerous research fields due to its proven performance in image recognition tasks are convolutional neural networks.

2.2.6.5 Convolutional Neural Networks

Convolutional neural networks comprise of multiple convolutional, pooling, and fully-connected layers that extract high-level features from images to make class predictions and classifications. A detailed description of the structure and function of the various CNN components can be found in section 2.4. Within the literature, there have been several applications of CNNs to automated defect detection. Atha and Jahanshai [32] explored the effects of color space and input image size on CNN performance in corrosion detection and found that the most robust parameters were an image size of 128 x 128 pixels and the RGB or YCbCr color space. It was also shown that CNN performance improved as image size increased due to smaller images being more susceptible to false positives. These results agree with convention as larger images generally result in increased, better-defined features for classification. In [33], Malekzadeh et al. used two state-of-the-art

CNN's as a feature extractor for an SVM classifier. The proposed algorithm achieved 96.38% accuracy on a small dataset of 12 images containing defects found on aircraft fuselage. Shen et al. [34] applied a CNN-based framework to perform semantic segmentation on borescope images of aircraft engines to detect cracks and burns. Semantic segmentation refers to pixel-wise classification of the image to localize defects in finer detail than image classification or detection. The CNN-based framework achieved a mean accuracy of 97.26% on the test data. Ramalingham et al. [35] developed a reconfigurable climbing robot to capture images of aircraft surfaces, similarly to [30]. A CNN designed to be deployed on mobile devices was used to detect defects and achieved an accuracy of 96.2% with an average confidence level of 97%.

2.3 Challenges and limitations of existing methods

There exist several challenges and limitations faced by the current state-of-art as it pertains to automated defect detection of aircraft surfaces:

1. Traditional heuristic-based methods such as texture and color analysis are difficult to apply to the wide range of defects found in aircraft. They are also susceptible to variations in the imaging environment and the size and shape of defects.

2. Deep learning methods are resistant to variations in environment and defects. They can be retrained for new inspection tasks without explicit programming. However, the reported performance of deep learning methods in the literature are insufficient for critical inspection tasks and heavily reliant on the generalization ability of a single learner. Ensemble models have the potential to further improve performance by combining the outputs of multiple networks and reduce the errors associated with the inherent bias or variance of any single model.

3. Very little exists in the literature regarding the relationship between false alarms, missed detections, and model accuracy when developing automated detection systems for aircraft maintenance. For inspection tasks, false alarms are tolerable while a missed detection may result in critical failure. A robust system aims to maintain high detection accuracy while minimizing missed detections, often at the cost of a low false alarm rate. Ensemble techniques can be applied in conjunction with threshold adjustment to manage and decrease the tradeoff between Type I and Type II errors.

2.4 CNNs

Convolutional neural networks are currently state-of-the-art in many image related applications, including automated defect detection. This section provides a brief description of the layers that comprise a CNN, the conventional training and evaluation process, and a review of existing architectures that are leveraged for this study. This study is limited to detect classification and therefore, techniques for segmentation or localization will not be discussed.

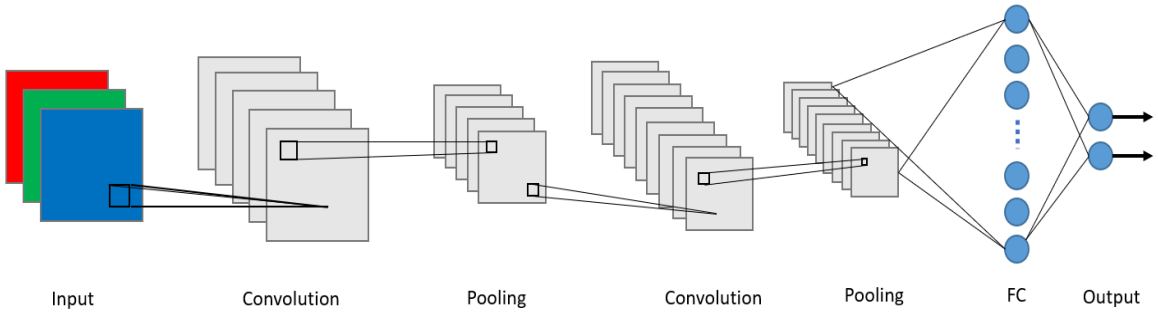

Figure 2.5: General representation of CNN architecture

2.4.1 Layers

2.4.1.1 Input Layer

The input layer contains the raw pixel values of the input image as a 3D volume of size *Width* x *Height* x *Depth*. The *Depth* parameter reflects the number of color channels present in the image. For example, grayscale images contain a single color channel while RGB images have three. As demonstrated in [32], other color spaces such as YCbCr and HSV are also viable. The 3D volume of image pixels is passed from the input layer into the first convolutional layer.

2.4.1.2 Convolutional Layer

Convolutional layers serve to extract features from the input and contain sets of learnable kernels that are spatially limited in width and height but extend through the full depth of the input volume. The weights of each kernel can be initialized with random or pre-trained values and can be updated in the training process of a CNN. The kernels are convolved across the width and height of the input volume and dot products are computed between the kernels and input to form a 2-dimensional map of the filter responses. These 2D maps are called activation maps and will be stacked to form the output volume of the convolutional layer. The indices of subsequent kernel calculations are determined by the stride length. CNNs have multiple convolutional layers and as the number of layers increase, the computational power required also increases. Figure 2.6 illustrates the convolution of a one-dimensional image with a *3*x*3*x*1* kernel.

Once all activation maps have been generated, a pre-defined activation function performs a mathematical operation that introduces non-linearity into the system. Without activation functions, CNNs would only be capable of linear mappings between inputs and outputs as the features being passed from layer to layer would be formed only through successive dot product operations. The most widely used activation function is the

Rectified Linear Unit (ReLU) defined by $f(x) = max(0, x)$ [37]. ReLU is popular as it is computationally inexpensive and has been shown to accelerate the convergence of gradient descent algorithms towards the global minimum of a loss function when compared to other activation functions. A common problem with ReLU functions arises when negative gradients are zeroed, and a large number of weights are not updated. In addition to ReLU, other activation functions include step, sigmoid, and tanh functions.

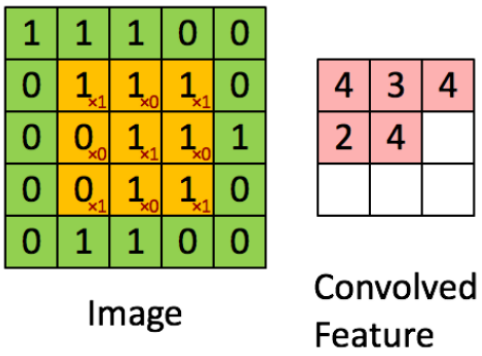

Figure 2.6: Convolution of a 5x5x1 image with 3x3x1 kernel from [36]

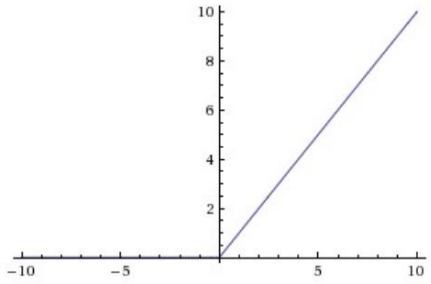

Figure 2.7: ReLU function from [37]

2.4.1.3 Pooling Layer

Pooling layers downsample the convolutional features in order to decrease computational costs and filter irrelevant features. The two primary methods are max and mean pooling. Max pooling returns the maximum value of the subarray operated upon by

the kernel while mean pooling returns the average value. Like in convolutional layers, the stride length determines the step size of the kernel. Figure 2.8 shows max and mean pooling with a *2* x *2 x 1* kernel and stride length of 2.

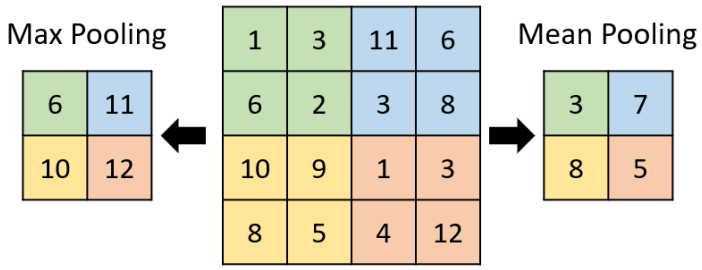

Figure 2.8: Max and mean pooling with 2x2x1 kernel and stride length of 2

A CNN contains multiple convolutional and pooling layers that are grouped together to form the *i*-th layer of the network. After the final pooling layer, the output is flattened into a 1-D vector for the fully-connected layer. Each value in the vector represents the probability that a certain feature belongs to a class.

2.4.1.4 Fully-Connected Layer

The fully-connected layer outputs the class scores in an *N* dimensional vector where *N* is the number of classes. The feature probabilities from the output vector of the last pooling layer are multiplied by the weights of the neurons in the fully-connected layer and passed through an activation function. These values are then sent to the output layer where a softmax algorithm will typically be used to make the final class prediction. There can be several fully-connected layers before the output layer of a CNN.

2.4.1.5 Training and Evaluation

Development of a deep learning model can be split into training and evaluation phases. Within the training phase, the weights of the model are incrementally updated until

an error threshold is met, or overfitting occurs. The data used to train the model is colloquially referred to as the training set. In the evaluation phase, the generalization performance of a model is evaluated against a testing set. The data within the training and testing sets are mutually exclusive and representative of real-world data. Further training and parameter tuning may be necessary if the model performs poorly.

2.4.1.6 Backpropagation

Backpropagation is the technique used to fine-tune the weights of a neural network by taking the partial derivative of the loss function with respect to any weight *w*. Loss functions are used to quantify the deviations between the model predictions and the target values. The two most commonly used lost functions are mean squared error (MSE) and cross-entropy loss. MSE is typically used for regression problems and can be calculated by averaging the squared differences between the actual and predicted values. Cross-entropy loss is usually used for classification problems and compares the predicted probability with the actual class output as seen in (1).

$$Cross\ Entropy\ Loss = -\sum_{c=1}^{N} y_c \log (p_c) \tag{1}$$

N is the number of classes, y_c is the binary value that represents if *c* is the correct class label, and *p* is the predicted probability.

Backpropagation can be separated into four different parts: 1) forward pass, 2) loss function, 3) backward pass, and 4) weight update. In the forward pass, the training image is passed through the whole network, resulting in a vector of class predictions. The loss function is used to quantity the error between the predictions and the actual class labels. In the backward pass, the derivative of the loss function is taken with respect to the weights

and the weights which contributed most to the accumulated loss are update via optimization functions. Backpropagation occurs after every iteration for the duration of the training.

2.4.1.7 Optimization Functions

Optimization functions are utilized within backpropagation to minimize the loss function by updating the weights of the model. The most widely used optimization function is stochastic gradient descent (SGD). SGD is a variant of gradient descent developed to overcome two challenges associated with the original function, namely local minima and saddle points. In Figure 2.9, weights initialized at point A would converge upon the local minimum and remain there. There is also a saddle point where a local minimum and maximum exist along perpendicular axes. Weights in this region would oscillate back and forth as the calculated loss increases in one direction but decreases in the other.

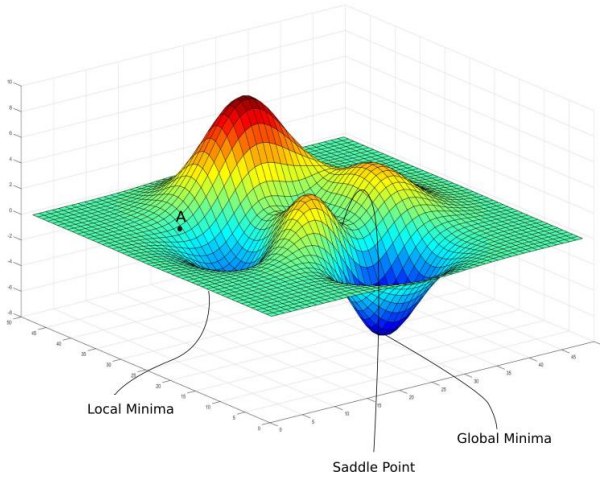

Figure 2.9: 3-D surface plot of loss function from [38]

While gradient descent calculates the exact value of the loss gradient across all training samples before updating, SGD updates by computing the gradient of a single randomly selected sample per iteration. This makes SGD computationally lighter and allows it to escape local minima as the next randomly selected point may point in a different

direction. A momentum technique for SGDs was developed to help it escape saddle points [39]. While generally faster than gradient descent, SGD is noisier, and the fluctuations may slow down the convergence. Equation 2 describes the update rule for SGD.

$$\nabla_{w_i} = -\alpha * \nabla L(w_i) \quad (2)$$

∇_{wi} is the change in weights, $\nabla L(w_i)$ is the gradient of the loss function calculated at any random *i*-th sample, and α is the learning rate.

2.4.1.8 Hyperparameters

Hyperparameters are the variables that determine how the CNN is trained and are defined before the first training iteration. These variables include the learning rate, momentum, number of epochs, and batch size.

Learning rate refers to the step size taken in the direction of gradient descent by the optimization function, as shown in (2). A learning rate that is too small will slow down convergence while one that is too large will cause the loss function to fluctuate at the minimum or overshoot it altogether [39]. A learning rate scheduler can be applied to adjust the learning rate during the training process. This allows for higher learning rates in early iterations to speed up convergence and lower rates further on in the training process to prevent fluctuations.

Momentum is a technique applied to accelerate gradients and reduce fluctuations associated with SGD by preserving the influence of the previous update direction in the next iteration [39]. The algorithm introduces a momentum factor into the calculations of gradients as shown in (3).

$$\nabla_{w_i} = -\alpha * \nabla L(w_i) + v * w_{i-1} \quad (3)$$

w_{i-1} is the previous weight update and v is the momentum factor. The momentum factor is less than 1 and typically between 0.5 and 0.9 for most applications. Epochs refer to a single cycle through the full training set during the learning process. Training CNNs requires more than a single epoch for a network to obtain the weights needed to make accurate predictions. The number of epochs is generally increased until the model begins overfitting, at which point the test accuracy begins decreasing even as training accuracy increases. Batch size refers to the number of training samples passed through the model per training iteration. There are multiple training iterations within an epoch. Batch size has an effect on convergence speed as the model weights are updated at the conclusion of each training iteration.

2.4.1.9 Transfer Learning

Very few CNNs are trained from scratch for a specific task due to the lack of sufficient datasets. Instead, it is common to take a model that has been trained on a prior task and apply it to a separate, but similar task. This technique is known as transfer learning. The success of transfer learning is dependent on the generality of the features learned in the first task, making CNNs pre-trained on expansive datasets such as ImageNet [40], which contains over a million images and 1000 classes, extremely popular in application. The two major transfer learning scenarios are feature extraction and fine tuning.

In feature extraction, the last layer of the pretrained CNN is replaced with a new fully-connected layer. The weights of the network are frozen except for the final layer and the model is retrained on the new dataset. Feature extraction assumes that the features extracted in the initial training set can be used make predictions on the second; however, this only applies when the datasets are sufficiently similar.

In fine-tuning, a CNN is initialized with the same weights as a pretrained network and then retrained on the new dataset. This allows the new CNN to converge quicker and with less data as the weights are not randomly initialized. It is also possible to freeze specific layers to further fine-tune the model.

2.4.1.10 *k*-Fold Cross Validation

Cross validation is a resampling procedure used to reduce bias and evaluate the performance of a CNN. The parameter *k* refers to the number of groups that a given dataset of images is split into. The procedure is described as follows:

1. Split the entire dataset randomly into *k* folds. The number of folds varies based on size of the dataset.
2. For each group *i* from 1 to *k*
    a. Take group *i* to be the testing set
    b. Combine the remaining groups into the testing set
    c. Train the model on the training set and evaluate on the testing set
    d. Log performance metrics and discard model
3. Average the metrics across all folds

A further modification can be made to *k*-folds cross validation to include a validation set in addition to the testing and training sets. The purpose of a validation set is to allow fine-tuning of the model parameters before evaluating on the testing set. The algorithm is adjusted such that one fold is removed from the process and the models are iterated through $k - 1$ folds following the procedure described above. The iterated models are evaluated on the validation set and can be retrained with new parameters before being exported. The

exported models are then evaluated on the removed fold, which serves as the common testing set. The modified *k*-fold algorithm reduces the risk of overfitting as model parameters can be fine-tuned to the validation set and then evaluated on the testing set. If the model is fit too closely to the validation set, there will be an evident decrease in evaluation performance.

2.4.1.11 Performance metrics and error representation

In addition to accuracy, model performance is evaluated on precision, recall, and F1 score. Precision expresses the proportion of positive identifications that were correct while recall describes the proportion of actual positives that were correctly identified by the model. F1 score is the harmonic mean of precision and recall and measures the balance between the two metrics.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{4}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negatives} \tag{5}$$

$$F1\ Score = 2\ x \frac{Precision * Recall}{Precision + Recall} \tag{6}$$

$$Accuracy = \frac{True\ Positive + True\ Negative}{Total} \tag{7}$$

Errors in CNNs are represented as a combination of bias, variance, and irreducible error as seen in Equation 8. Due to the stochastic nature of CNNs and their datasets, independent training sessions will result in models with a range of predictions. Bias

captures the average difference between the model prediction and the true class while variance captures the variability of a model prediction for a given point. Irreducible error is a result of the inherent noise in the classification task that cannot be reduced by any model.

$$Error = Bias^2 + Variance + Irreducible\ Error \quad (8)$$

There exists a tradeoff between bias and variance that can be managed during the training process through hyperparameter tuning or dataset selection. For example, in cases of overfitting where the model has low bias on the training set but high variance in the testing test, it is common to lower the learning rate and number of epochs for subsequent trainings. Complex models typically struggle with low bias and high variance while less complex models are susceptible to high bias and low variance.

2.4.2 Existing Architectures

CNNs were shown to be successful in image recognition tasks in 1989 [41] but due to the computational requirements, there were minimal advancements until the creation of the ImageNet challenge in 2009 [40]. This coincided with the development of GPUs with sufficient bandwidth and parallel processing for rapid feedback of trained models. CNNs have since demonstrated classification performance greater than humans on the ImageNet challenge; however, CNN performance is more volatile and declines rapidly with decreasing signal-to-noise ratio under image degradations like additive noise [42]. The architectures used in this study are VGG Net, GoogLeNet, and ResNet.

2.4.2.1 VGG Net

The VGG architecture [43], developed in 2014, showed that the depth of a network is a critical component in achieving better classification accuracy with CNNs. It also reduced the number of parameters within the convolutional layers by using a fixed kernel size of *3* x *3* to replace previously variable kernel sizes. For example, a *7* x *7* kernel can be replaced by three *3* x *3* kernels with a stride length of 1. This reduces the number of parameters by 44.9 % as a single *7* x *7* requires $7^2 = 49$ parameters and three *3* x *3* kernels only require $3(3^2) = 27$ parameters. VGG networks of varying lengths ranging from 11 – 19 layers were presented, and the architecture was the runner up in the 2014 ImageNet Challenge with a top-5 error rate of 7.3%.

2.4.2.2 GoogLeNet

GoogLeNet [44] was presented at the same ImageNet Challenge as VGG Net and took first place with a top-5 error rate of 6.67%. It further reduced the number of parameters from 138 million in VGG-19 to 6.8 million. GoogLeNet introduced the concept of inception layers where kernels of sizes *1* x *1, 3* x *3,* and *5* x *5* are applied in parallel to the output from the previous layer. Doing so allowed the network to handle large variations in the size of salient features within the input image. For example, when training a model to classify cars, the training image may be cropped such that the car occupies most, some, or only a tiny portion of the total area. Large kernels are suitable for features that are globally distributed while smaller kernels are preferred for locally distributed features. The addition of inception layers that contain multiple kernel sizes allow GoogLeNet to detect features of varying sizes to improve classification performance. A *1* x *1* kernel is added before the *3* x *3* and *5* x *5* convolutions to reduce the depth parameter. Figure 2.10 shows the dimension reducing inception module presented in [44]. GoogLeNet contains 22 layers, three more than the largest VGG model, but was able to significantly reduce the computation cost and time required to train due to the parameter reduction of the *1* x *1* kernel.

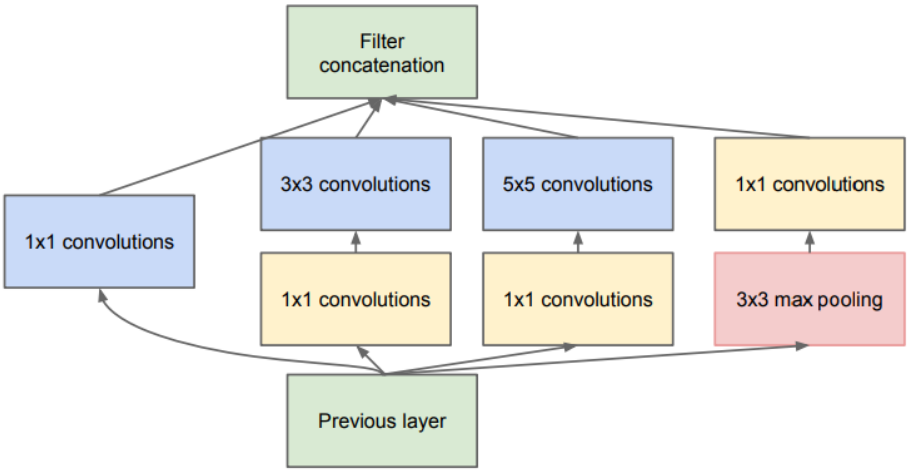

Figure 2.10: Inception module from [44]

2.4.2.3 Residual Networks (ResNet)

Residual Networks [45] were designed to combat the vanishing gradient problem experienced by other neural networks as the partial derivative of the loss function calculated in backpropagation becomes insignificant in the initial layers. As the number of layers increased, accuracy becomes saturated and degrades rapidly. ResNet makes use of residual blocks that map the input *x* to some output *F(x) + x* as it is simpler to optimize the model by treating *F(x)* as some delta in *x* rather than a function. Figure 2.11 illustrates the basic residual block.

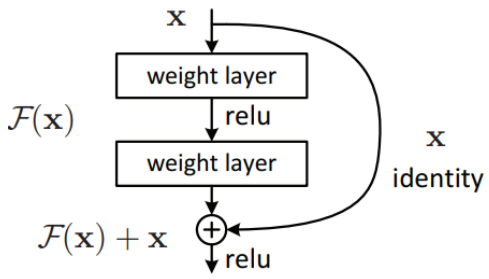

Figure 2.11: Basic residual block from [45]

ResNet also makes use of batch normalization [46] to standardize the inputs to the network by scaling the data to have a mean of zero and standard deviation of one. Batch normalization speeds up training and reduces overfitting due to its regularization effects. A 152-layer ResNet network was presented at the 2015 ImageNet challenge and achieved a top-5 error rate of 3.57%, becoming the first CNN to exceed human classification ability.

2.5 Ensemble Learning

An ensemble consists of a set of individually trained machine learning models whose outputs are combined into a single predictive model. Prior research has shown that ensembles improve classification performance and reduce variance as compared to any single classifier within the ensemble [47]. The classifiers which comprise an ensemble are referred to as base learners. For the base learners, each training phase will result in a model with a different set of weights that correspond to different methods of generalizing about the training data. A combination of the different generalization methods is likely to reduce error in the final ensemble, given sufficient diversity in model errors. Performance also relies on structural diversity within the base learners, as models which share structures are likely to make correlated errors, decreasing the overall error reduction provided by ensembles [48]. Three of the most common techniques for creating ensembles are bagging, boosting, and stacking.

Bagging [49] stands for bootstrap aggregation and is the simplest of the three methods. The algorithm uses datasets generated through random sampling with replacement to train individual base learners whose results will be combined through averaging or majority vote to form the final prediction. Averaging is typically used in regression problems while majority vote is used for classification. Bagging is effective in reducing variance and maintains bias, resulting in an overall decrease in generalization error.

Boosting comprises of a family of algorithms that improve the performance of weak learners through successive training. Weak learners are models that are close in performance to random guessing. One of the most widely used boosting techniques is Adaboost [50] where classifiers are trained sequentially on weighted data until performance has saturated on the training set or the maximum capacity of models is reached. The weights of misclassified items are increased while the weights of correctly classified items are decreased. CNNs are rarely used in boosting as they are too computationally expensive. Boosting results in a large number of sequentially trained models, making it impractical to use CNNs. Furthermore, boosting is more effective on models that underfit whereas CNNs typically deal with overfitting. Underfitting occurs when the model has too few features, making it unable to provide accurate predictions on complex inputs.

Stacking combines multiple classification models by training a second level model on the outputs of the base learners [51]. The second level model or *meta-learner* can conditionally weigh the input predictions based on context, improving ensemble performance. Typical meta-learners used in stacking include logistic regression, random forests, and SVMs. Unlike bagging, which requires multiple diverse models generated by a single learning algorithm, stacking performs well with two or three base learners [52]. Furthermore, stacking is better suited for base learners with a wide range in performance. Stacking can be extended to many levels, though constraints on computational power usually make large stacking networks impractical. Figure 2.12 shows the structural differences between a single model CNN framework and a stacked ensemble consisting of CNN base learners and a logistic regression meta-learner.

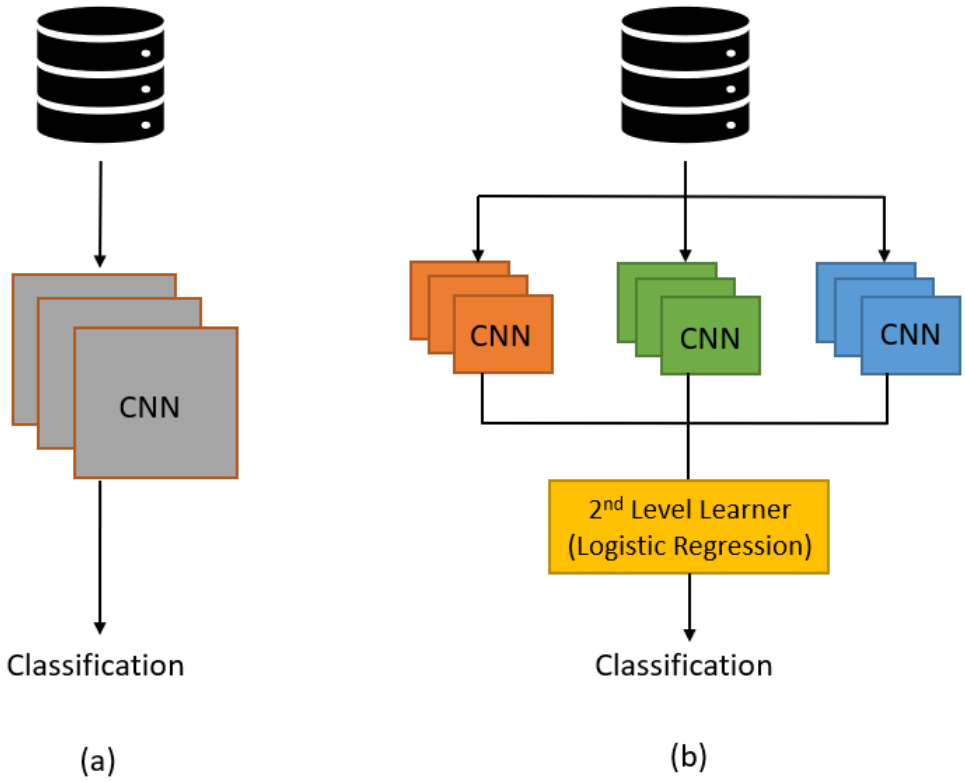

Figure 2.12: Components of (a) single model CNN and (b) stacked ensemble

CHAPTER 3. METHODOLOGY

The goal of this study is to develop an ensemble framework for the automated defect detection of surface defects in aircraft. The classes of defects present in the dataset include cracks, gouges, and corrosion. Stacking will be used to construct the ensemble with CNNs as the base learners and a logistic regression model as the meta-learner. Performance of the generated ensembles will be compared to the current state-of-the-art in automated aircraft inspection.

3.1 Data

3.1.1 Data Acquisition

The data consists of images gathered from a borescope inspection process for C130 propeller blades at Robins Air Force Base. The borescope inspection is the first visual inspection within a larger repair and overhaul process. Currently, the propeller blades are mounted to an inspection frame whereupon a Fanuc LR Mate 200iD robotic arm indexes a borescope along the interior surface of the bore. A CCD camera is configured to capture images from the borescope at set depth and angle intervals. A total of 455 images are captured per blade. Two trained operators scan the images for defects and note the locations of identified defects on a grid that is then passed along to the next step in the overhaul process. The current process takes approximately two hours and four blades are inspected per shift. Future production goals aim to increase the throughput to eight blades per day, necessitating the development of an automated inspection system that can decrease inspection times and augment the abilities of the trained operators. A fluorescent penetrant

inspection down the line also relies heavily on visual inspection and can be similarly improved by an automated system.

3.1.2 Data Preprocessing

The borescope images were cropped to a size of *224* x *224* x *3* pixels to match the sizes of the input layers of the CNNs used in this study. There is no requirement for specific image sizes in CNNs, however, images of size *224* x *224* or *256* x *256* have become convention due to the tradeoff between image size, network size, and number of parameters. Larger images are noisier, requiring networks to be have more convolution layers. This in turn requires larger datasets to tune the additional parameters. For smaller images, the features extracted in the first few layers are lost within the network due to the dimensional reduction of the pooling layer. This compromise between size and network complexity is why CNNs such as VGG and ResNet use square images of size *224* x *224*. Figure 3.1 illustrates the image cropping process.

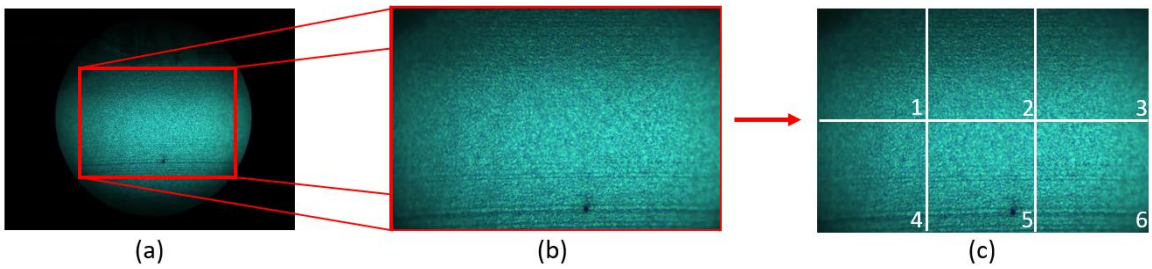

Figure 3.1: Image cropping - (a) original image 1280 x 960 pixels, (b) functional region 672 x 448 pixels, and (c) CNN input layer size 224 x 224 pixels

After cropping, the images are classified by a subject matter expert into binary classes, *defect* and *defect-free*. The classified images were placed in separate sub-directories, each containing 300 images. The images were left in the RGB color space as it was shown in

[32] that there were no substantial improvements associated with alternative color spaces. The variations in lighting and defect geometry found in the images can been seen in Figure 3.2.

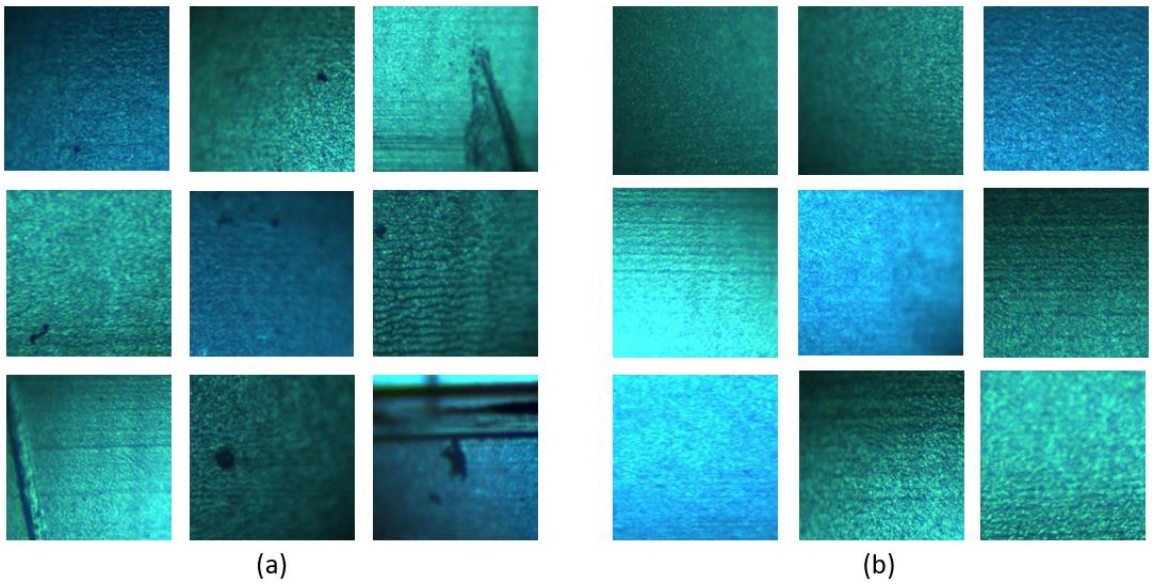

Figure 3.2: Sample images – (a) defect images, and (b) defect-free images

3.1.3 Training/Validation/Testing Split

A modified cross validation algorithm with a holdout fold was used to split the data into training, validation, and testing sets. The 600 total images were randomly split into six folds. One fold was held out to serve as the testing set and a standard 5-folds cross validation was applied to the remaining folds as seen in Figure 3.3. Each fold contains an equal distribution of *defect* and *defect-free* images. 5-folds cross validation resulted in five training and validation pairs, denoted *Datasets 1* through *5*, that are used to train and tune the CNNs. The testing set containing 50 *defect* images and 50 *defect-free* images is used to compare the performance between the ensembles and the base learners.

Figure 3.3: Splitting of dataset into training, validation, and testing sets

3.2 Model Selection and Training

3.2.1 Base Learners

The base learners used in this study consist of CNNs that have been pretrained on the ImageNet dataset. The selected architectures are *ResNet18*, *ResNet34*, *ResNet50*, *VGG11_bn*, and *GoogLeNet*. The three ResNet networks were chosen to represent diversity in depth while *VGG11_bn*, and *GoogLeNet* add diversity in structure. Batch normalization has been applied to the VGG-11 network to improve accuracy, resulting in *VGG11_bn*. These CNNs have been previously applied to automated defect detection in the literature. VGG networks are found in [32] and [33]. [52] uses both VGG and ResNet architectures. Each network was trained twice on *Datasets 1* through *5* to generate a total of ten models per network. For the remainder of this study, network will be used to refer to the general CNN architecture while model will be used to denote individual trained samples.

Data augmentation in the form of a random horizontal flip was performed on the training samples to increase the size and diversity of the training set. This technique is widely used in neural network applications where generating new data is difficult. Other common data augmentation techniques include padding, cropping, and rotations. The augmentations used in this study are limited as initial testing had shown that additional techniques beyond the initial horizontal flip had little to no effect on model performance. These additional techniques include vertical flips, brightness and contrast adjustment, and Gaussian blurring.

The hyperparameters defined in the training were determined by evaluating the training and validation error of the models over successive runs.

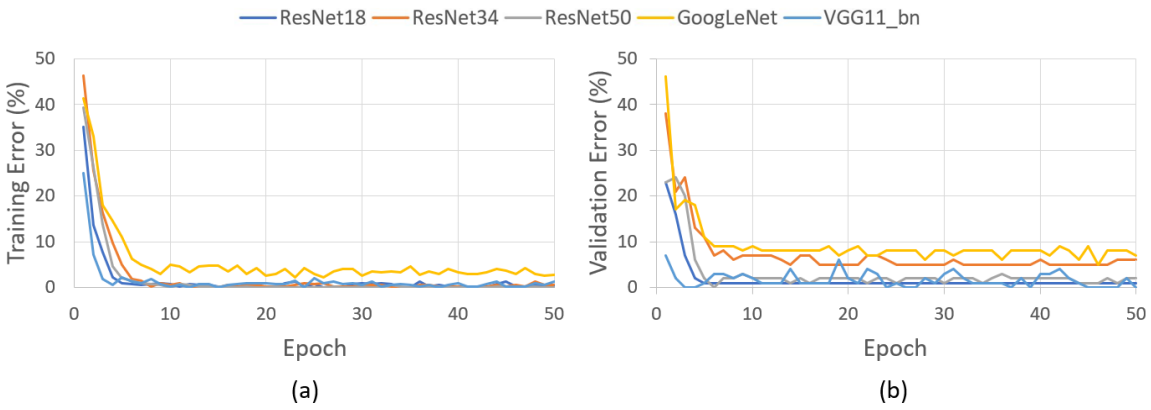

Figure 3.4: Training and validation error rates by epoch for different CNN architectures – (a) training error by epoch, and (b) validation error by epoch

Figure 3.4 shows the training and validation losses of the five networks over 50 epochs. The number of epochs was set to 25 for subsequent trainings as the training and validation error had converged by that point. Batch size was set to 25 for all of the networks except *VGG11_bn* which has the highest number of trainable parameters. A batch size of 5 was used instead due to memory constraints on the training device. A cross entropy loss

function was used in conjunction with an SGD optimizer with momentum. Momentum was set to 0.09. The learning rate was set at 0.001 for the initial training iteration but a scheduler is specified to decrease it a factor of 10 every 7 epochs. As the models had been pretrained on the ImageNet dataset, fine-tuning was used to apply the models to the borescope images. The final classification layer was replaced and all of the model's weights were updated during training.

During training, the validation error of the model was tracked between iterations. A copy of the model weights that resulted in the lowest error is stored separately and not updated through backpropagation. As the best weights do not always occur in the last training iteration, this technique allows for the best fitting model to be exported from training.

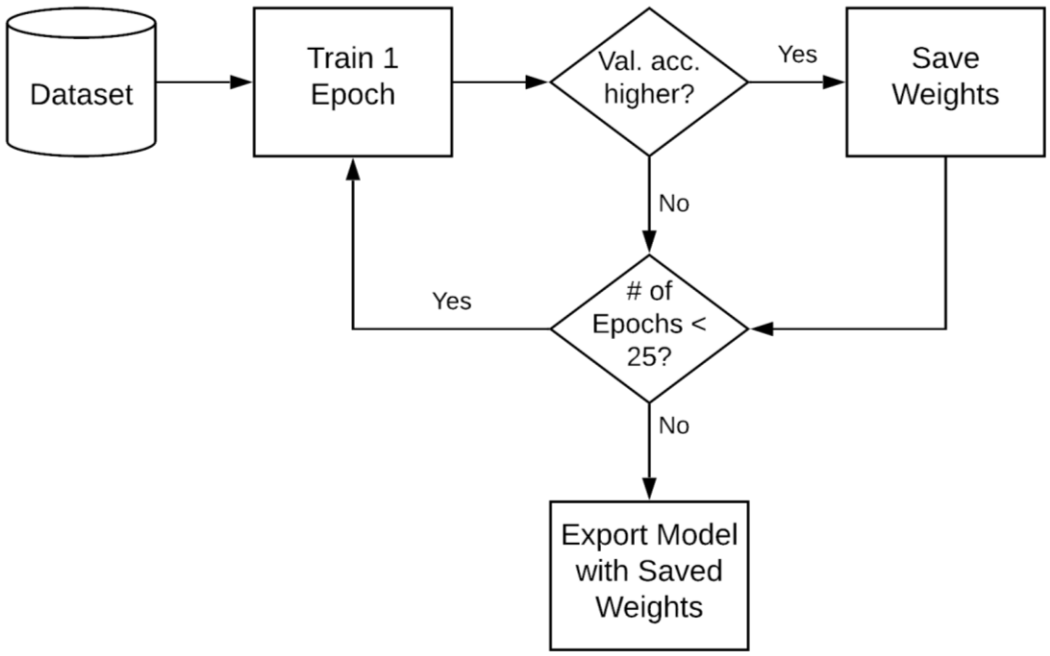

Figure 3.5: Flow chart of CNN training process

3.2.2 Ensemble Construction

Stacked ensembles were constructed from the trained CNNs and a logistic regression was selected as the meta-learner. Each ensemble consists of three base learners and a single meta-learner as shown in Figure 2.12. The logistic regression is trained on the predictions made by the base learners on the testing set. Three types of ensembles were constructed to evaluate the effects of depth and structure diversity on error reduction. *Ensemble A* is the sole homogenous ensemble and consists of three *ResNet18* models. *Ensemble B* is composed of *ResNet18*, *ResNet34*, and *ResNet50* models and explores diversity in model depth. *Ensemble C* is composed of *ResNet18*, *GoogLeNet*, and *VGG11_bn* models and explores diversity in model structure.

Table 2: Different ensemble structures

Ensemble	Base Learners
Ensemble A	3x ResNet18
Ensemble B	ResNet18, Resnet34, Resnet50
Ensemble C	ResNet18, GoogLeNet, VGG11_bn

3.2.3 Tools

PyTorch [53] is an open source machine learning library developed by Facebook's AI Research lab and was used to train the base learners. The CNNs were recreated within the PyTorch framework and reevaluated against the ImageNet challenge to establish a baseline for expected performance. The results are shown in Table 3. The differences in ImageNet performance from that reported in the literature is due to updates to the ImageNet

challenge that added more image classes and an inexact replication of the networks. It is difficult to determine the exact hyperparameters used to achieve the results reported by the original authors.

Table 3: Error rates of PyTorch models on ImageNet challenge from [53]

Network	Top 1 Error (%)	Top 5 Error (%)	Number of Parameters
ResNet18	30.24	10.92	11.7M
ResNet34	26.70	8.58	21.8M
ResNet50	23.85	7.13	25.6M
GoogLeNet	30.22	10.47	6.8M
VGG11_bn	29.62	10.19	132.9M

The logistic regression meta-learner was trained using a Python package from scikit-learn [54]. Training and evaluation were performed on a computer with a NVIDIA Geforce GTX 1050Ti GPU, Intel i7-8750H CPU, and 8GB of RAM.

CHAPTER 4. RESULTS AND DISCUSSION

This section details the results of the trained CNNs and stacked ensembles on the borescope image set. The confusion matrix for this study is defined in Figure 4.1. The null hypothesis is that the image contains a defect. False alarms are defined as Type I error and missed detections are defined as Type II error.

H_o: The image contains a defect

		Defect Presence	
		H_o is true	**H_o is false**
Model Classification	**Accept H_o**	True Positive	False Positive Type I Error
	Reject H_o	False Negative Type II Error	True Negative

Figure 4.1: Confusion matrix

4.1 CNN Results

The selected CNN networks were evaluated on the testing set to obtain a performance benchmark. Means and standard deviations for the accuracy, precision, recall, and F1 scores of the models are presented in Table 4. Ten models were trained per network and the Shapiro-Wilks test was used to determine that the evaluation results were non-normal. Using the Mann-Whitney U-test with a significance value of 0.05, the *ResNet18* network was found to have a greater mean accuracy, recall, and F1 score than the other networks, outside of *ResNet50* which had similar performance as seen in Table 5.

VGG11_bn achieved the highest precision but its overall accuracy suffered due to its high recall.

Table 4: Performance of different single-model CNN networks on test images

Network	Accuracy (%)		Precision (%)		Recall (%)		F1 Score (%)	
	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev
ResNet18	98.10	0.99	98.20	1.48	98.03	1.60	98.10	0.99
ResNet34	96.80	1.40	97.40	1.90	96.26	1.70	96.82	1.40
ResNet50	97.40	0.97	97.80	1.75	97.06	1.62	97.41	0.97
GoogLeNet	93.60	1.96	96.40	1.58	91.43	3.38	93.81	1.85
VGG11_bn	95.00	2.36	98.40	1.35	91.52	4.00	95.25	2.14

Table 5: One-tailed Mann Whitney U-test

H_0: Group 1 ≤ Group 2, H_1: Group 1 > Group 2 α = 0.05, N = 10 for Group 1							
Group 1	N_1	Group 2	N_2	Accuracy		F1 Score (%)	
				Z	*p*	Z	*p*
ResNet18	10	ResNet34	10	2.103	0.018	1.797	0.036
ResNet18	10	ResNet50	10	1.349	0.088	1.033	0.151
ResNet18	10	GoogLeNet	10	3.770	8.15E-5	3.753	8.73E-5
ResNet18	10	VGG11_bn	10	2.926	0.002	2.466	0.007

The large variance in the recall of *VGG11_bn* is often a sign of overfitting. Due to memory constraints, the network was trained with a batch size of 5, which means that its weights were updated 5 times as frequently as the other models which used a batch size of 25. However, the network simultaneously demonstrates high precision, making it unlikely that the model is simply overfit to the training set as overfitting would result in a reduction in both precision and recall. Rather, it appears that the confidence thresholds that determine

the final class prediction need to be adjusted to better balance the Type I and Type II error rates. Confidence thresholds refer to the threshold above which class probabilities outputted from the fully connected layer are sufficient to make predictions. For binary classifiers like those in this study, the threshold is usually set at 50%. Reducing the confidence threshold would result in fewer missed detections, raising the recall and lowering the precision of the model.

The performance of the base learners is comparable to the performance of single model inspection frameworks found in the literature [33 – 35]. Since ResNet had outperformed the other two networks on the ImageNet challenge, it was unsurprising that *ResNet18* and *ResNet50* were the best performing in this task. However, *GoogLeNet*, which had similar reported performance as *VGG11_bn* in the ImageNet challenge, struggled and had the lowest average scores amongst the networks. This may be due to the nature of the inception modules within *GoogLeNet* that are designed the detect salient features of varying distribution. In the borescope image set, the defects are typically small and highly localized, rendering the larger convolutional kernels ineffective. These kernels would obfuscate the feature extraction, resulting in the poor performance of *GoogLeNet* relative to the other networks. A sample of the predictions can be seen in Figure 4.2.

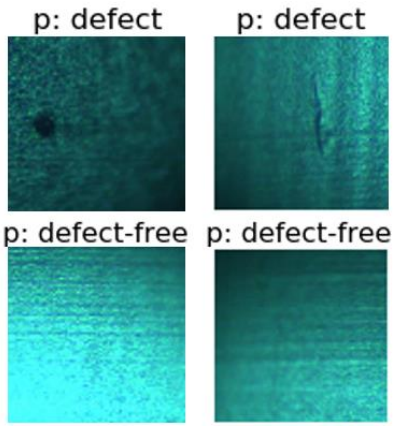

Figure 4.2: Sample of predictions made by a ResNet18 model

4.2 Ensemble Results

The stacked ensembles specified in Table 2 were constructed from the library of 50 trained base learners. To highlight the effects of base learner selection on ensemble performance, 30 ensembles were constructed using random selection of base learners with replacement while another 30 were constructed with manually selected base learners. Two additional ensembles were created to demonstrate an ensemble's ability to selectively minimize Type I or Type II error without adjusting confidence thresholds. Each ensemble contains a unique combination of base learners. The results of the randomly constructed ensembles can be found in Table 6.

Table 6: Performance of randomly constructed stacked ensembles on test images

Ensemble	Accuracy (%)		Precision (%)		Recall (%)		F1 Score (%)	
	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev
Ensemble A	98.70	0.82	98.61	0.96	98.80	1.40	98.70	0.83
Ensemble B	98.20	0.63	98.21	0.63	98.20	1.48	98.20	0.64
Ensemble C	98.10	0.74	98.21	1.10	98.00	1.33	98.10	0.75

Again, a Mann-Whitney test was used to compare the means. While the performance of *Ensemble A* appears higher, there is no statistically significant difference between the ensembles. This is interesting as the base learners which comprise *Ensemble C* performed significantly worse, on average, than those that comprise the other two ensembles. The larger error reduction can be attributed to the addition of the logistic regression and sufficient diversity in individual model errors. Diversity in model errors can be defined for the purposes of this study as the number of images misclassified by at least two of the base learners. The impact of a suitable meta-learner is further evident in the results shown in Table 7 where the logistic regression is replaced with a bagging operator.

Table 7: Performance of bagging ensembles

Ensemble	Accuracy (%)		Precision (%)		Recall (%)		F1 Score (%)	
	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev
Ensemble A	98.50	0.71	98.61	0.96	98.40	1.27	98.50	0.71
Ensemble B	97.80	0.63	98.19	0.64	97.40	1.35	97.79	0.65
Ensemble C	97.30	1.16	98.37	0.86	96.20	1.99	97.26	1.19

There is a notable decrease in *Ensemble C*'s performance under the bagging operator with the difference in recall being the most significant. Additionally, the bagging ensemble had a lower overall accuracy than one base learner,*ResNet18*, due to the poor results of the other two base learners, *GoogLeNet* and *VGG11_bn*. In the stacked ensemble, the logistic regression was able to maintain overall accuracy due to the presence of learned coefficients that increased the weights of the predictions made by the *ResNet18* model. These results are consistent with the literature and further demonstrates the advantages of stacked ensembles over bagging ensembles for base learners with a wide performance range.

Table 8: Performance of manually constructed stacked ensembles on test images

Ensemble	Accuracy (%)		Precision (%)		Recall (%)		F1 Score (%)	
	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev
Ensemble A	99.80	0.42	99.61	0.83	100.00	0.00	99.70	0.48
Ensemble B	99.50	0.527	99.22	1.01	99.80	0.63	99.5	0.52
Ensemble C	99.60	0.52	99.80	0.62	99.40	0.97	99.6	0.52

The results of the 30 manually constructed ensembles are shown in Table 8. The base learners were selected to maximize error diversity. Using the Mann-Whitney test, it is found that all of the manually constructed ensembles had a greater mean accuracy, precision, recall, and F1 score than their randomly constructed counterparts. Many of the evaluation results approached perfect performance, however, given the limited size of the testing set, this is not likely to be reflective of true operational performance. There may be geometric variations and rare defect types found in the propeller blade inspection process that are not present within the dataset and exceed the limits of the ensembles' generalization ability. As with many deep learning approaches, the ensembles in this study can be continually trained as additional data becomes available without the need for a complete redesign of the system.

The performance differences between the manually and randomly constructed ensembles can be best attributed to error diversity. In the manually ensembles, the number of correlated errors was limited to zero or one, whereas in the randomly constructed, that number ranges between zero and five. Due to the limited number of base learners and input samples, the logistic regression performs poorly in cases where multiple learners misclassify the same image.

In addition to improving performance, stacked ensembling can also be used to manage the tradeoff between Type I and Type II errors. Table 9 and Table 10 show two ensembles with perfect performance in recall and precision respectively. While some ensembles in this study achieve 100% accuracy, precision, and recall, sub-optimal combinations are used to demonstrate this technique.

Table 9: Ensemble B with 100% recall on test images

Network	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
ResNet18	97.00	98.00	96.08	97.03
ResNet34	96.00	96.00	96.00	96.00
ResNet50	97.00	98.00	96.08	97.03
Ensemble B	**99.00**	**98.04**	**100.00**	**99.01**

Table 10: Ensemble C with 100% precision on test images

Network	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
ResNet18	97.00	98.00	96.08	97.03
GoogLeNet	91.00	94.00	88.88	91.37
VGG11_bn	96.00	96.00	96.00	96.00
Ensemble C	**97.00**	**100.00**	**94.00**	**96.91**

For aircraft inspection, where false alarms are tolerable, ensembling can be used to eliminate missed detection while maintaining a high level of precision as shown in Table 9. The same technique can also be used to eliminate false alarms as shown in Table 10. Given sufficient computational resources, the ensembles in Tables 8 and 9 can be stacked to eliminate both Type I and Type II errors on the testing set. Traditionally, the minimization of Type I or Type II error is done by adjusting the confidence threshold of

the CNN. This does not reduce the total error within the model and only serves to shift the error into either false negatives or false positives. Ensembles are successful in reducing the total error resulting from the bias-variance tradeoff within the system, increasing the level of precision found at perfect recall for critical inspections. Threshold adjustment can still be applied to the meta-learner to further fine-tune the error tradeoff.

Precision-recall curves were generated for the base learners and randomly constructed ensembles to illustrate the effects of confidence thresholds on model performance. The randomly constructed ensembles were selected as many of the manually constructed ensembles already achieve 100% accuracy on the testing set. The confidence threshold was adjusted from 0 to 100 percent at intervals of 5 to create the curves shown in Figure 4.3. For the ensembles, only the confidence threshold of the logistic regression was altered. The results presented are the average across 10 samples.

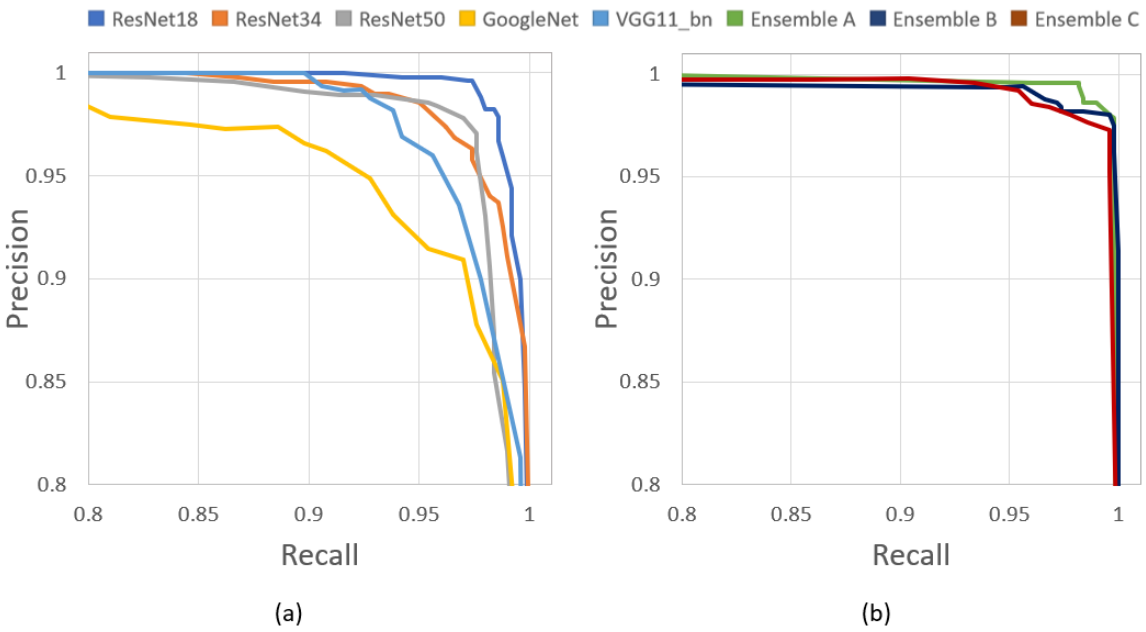

Figure 4.3: Comparison of precision-recall curves for CNNs and ensembles

The optimal point on the curve is located in the upper right corner, at the point of perfect precision and recall. For the purposes of aircraft inspection, where perfect recall is desired, Ensembles *A*, *B*, and *C* achieve precisions of 98.06%, 98.03%, and 97.28% respectively while maintaining a recall greater than 99.5% on the testing set. In comparison, *ResNet18,* attains the highest precision for base learners of 89.95% at 99.6% recall. The class probabilities for the ensembles were concentrated between 90 - 100% whereas due to the complexity of the base learners, there is a larger variance in their probabilistic outputs. The high accuracy of the base learners, sufficient diversity in model errors, and limited input features for the logistic regression result in the low variance in ensemble predictions.

For this application, the diversity in model depth and structure found in *Ensembles B* and *C* did not result in a measurable difference in classification performance relative to the homogenous *Ensemble A.* Based on the literature, it was expected that increasing diversity in model depth and structure would result in increased error diversity and better classification performance. However, in this study, these effects were masked by the accuracy and performance of *ResNet18*. *ResNet18* achieved the highest mean accuracy, recall, and F1 scores among the base learners and as a result, was more heavily weighted by the logistic regression meta-learner. It was also able to produce sufficient error diversity through different training iterations on different datasets. The weighted influence of *ResNet18* on the ensemble results and its lack of correlated errors resulted in the lack of differentiation between ensemble performance. Effects of model diversity on correlated errors is more apparent in the analysis of individual misclassified images in Section 4.4.

4.3 Comparison of CNNs and ensembles

The Wilcoxon test with significance level of 0.05 was used to compare the results of the base learners and ensembles as the distribution are non-normal and dependent. The randomly constructed ensembles were compared first to its base learners. *Ensemble A* only demonstrated an improvement in mean F1 score over *ResNet18*. While the mean accuracy, precision, and recall were numerically higher, the values were insufficient to establish statistical significance. *Ensemble B* showed improvement in mean accuracy, recall, F1 score over *ResNet34* and improvement in mean accuracy over *ResNet50*. There was no significant difference with the results of *ResNet18*. *Ensemble C* has a greater mean accuracy, recall, and F1 score over *GoogLeNet* and *VGG11_bn*. Again, there were no significant differences with the results of *ResNet18*.

The Wilcoxon test was used again to compare the manually constructed ensembles with their base learners. The results of the Wilcoxon test can be found in Appendix B. It was found that all of the ensembles had greater mean accuracy, precision, recall, and F1 score than their respective base learners.

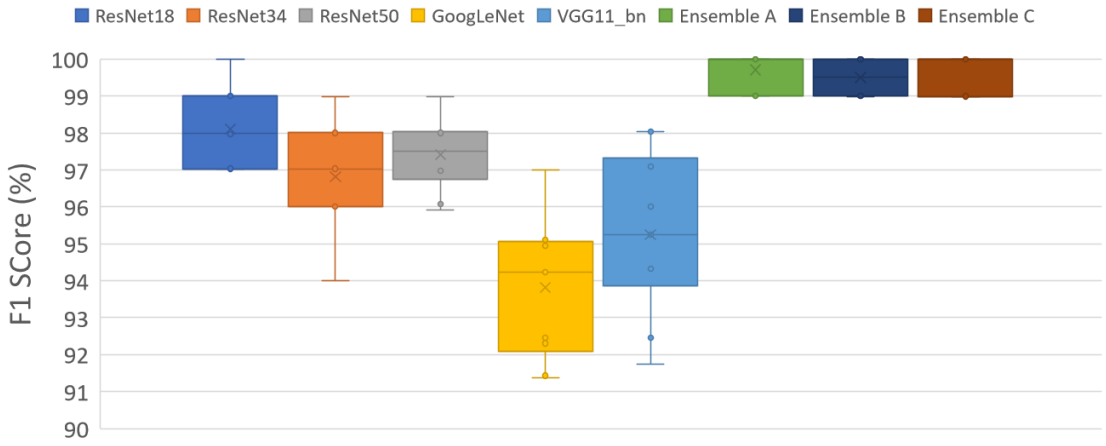

Figure 4.4: F1 scores of different CNNs and manually constructed ensembles

Classification accuracies of 96.38% in [33] and 97.27% in [34] are reported for CNN-based detection systems in the literature. The ensembles presented in this study achieve greater mean accuracy, precision, recall, and F1 scores than the current state-of-the-art; however, the lack of a common testing and training set makes a direct comparison difficult. Nonetheless, the results show that ensembles can augment the classification capabilities of single model CNNs through stacking and it's expected that this methodology can be applied to other inspection datasets to similar effect.

4.4 Model limitations and assumptions

The primary limitation of the stacked ensembles used in this study is related to the generalization abilities of the base learners and the number of correlated errors. Figure 4.5 shows the most commonly misclassified samples within the testing set for each respective base learner.

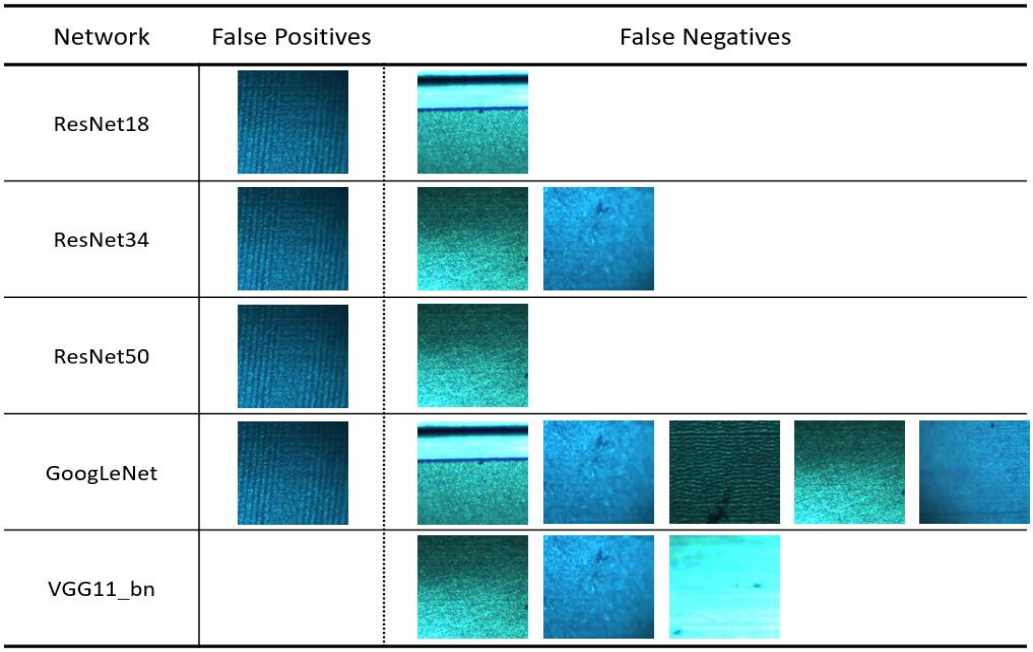

Figure 4.5: Most commonly misclassified images for each CNN network

There was a single correlated false positive across four of the five base learners. Given the "black box" nature of CNNs, it is difficult to provide an explanation for the outputs with certainty; however, it is likely that the rough surface present in the image lead the models to falsely classify it as a *defect* image. *VGG11_bn* was uniquely able to correctly classify this image due to its high sensitivity for false positives as shown in Table 4. The commonly missed false negatives share two main characteristics: 1) The defects are small and poorly defined. 2) The defects are located at the fringes of the image. These types of defects disappear within the convolutional and pooling layers as the kernels reduce the dimensionality of the features, making it difficult for CNNs to detect them. There are also images that have excessive glare due to lighting adjustments made by the operators and again, the defects are obfuscated. In contrast, Figure 4.6 shows a sample of correctly classified images by the *ResNet34* model. The selected images are reflective of the performance of the other models in this study. The true positive images contain defects of varying sizes that are easily differentiable from the surroundings while the true negatives contain defect-free surfaces under different levels of illumination. It can be seen that small variations in surface finish do not affect the classification results.

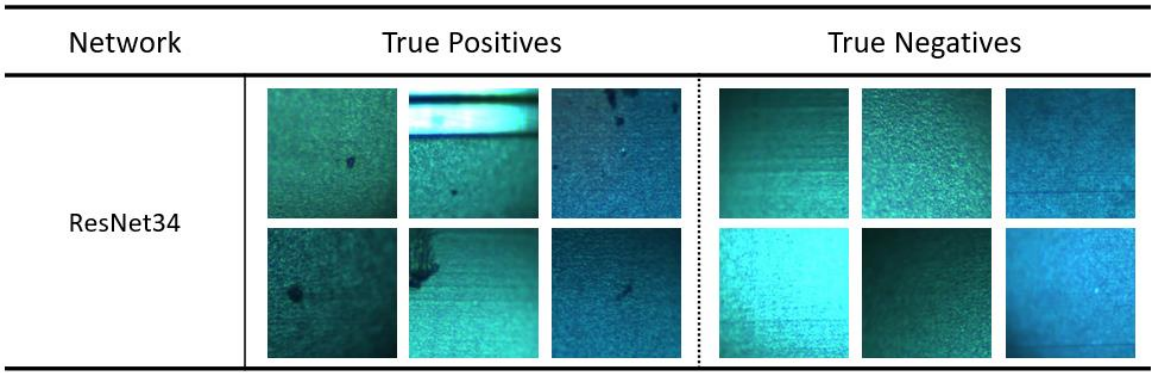

Figure 4.6: Sample of correctly classified images

Within each network, there are individual models which correctly classify some or all of images present in Figure 4.5 while misclassifying others. This variation in predictions is best attributed to the differences in training and validation sets as the hyperparameters are constant within each network. Despite the lack of significant differences in ensemble performance, the effect of model diversity and depth on correlated errors can be seen in the commonly misclassified images for each network. For example, the false negative for *ResNet18* in Figure 4.5 was misclassified by six of the ten trained models while in *ResNet34*, it was only misclassified by two. This unlikely to be a result of dataset or hyperparameters as only one of the two training instances for *ResNet34* shared a common dataset and the hyperparameters were the same for both networks. The additional depth of *ResNet34* over ResNet18 lead to a different distribution of false negatives. Similar examples can be found for the other networks. Stacked ensembles do not serve to improve the generalization performance of individual base learners; rather they try to find the optimal combination of inputs to generate the correct output. Therefore, the selection of base learners with a diverse distribution of model errors is critical to the performance of a stacked ensembles and models that have insufficient error diversity experience rapid degradation in performance.

Another limitation exists in the selection of the logistic regression model as the meta-learner. Logistic regressions are typically only used for linearly separable binary class predictions and other algorithms are more suitable for multi-class applications. Furthermore, it is susceptible to overfitting and being overly confident in its predictions, particularly in applications with a small dataset. The training set for the logistic regression only contained 100 samples whereas the base learners had 400 samples in their training

sets. While this did not pose in issue in the testing set, it is difficult to confidently determine if the model has been overfit without additional data. This lack of confidence can also be extended to the base learners and additional testing for both are required before this framework can be placed in production.

The composition of the dataset is crucial to the performance of the ensembles. This study makes the following assumptions regarding the dataset:

1. The images are representative of the true defect conditions and capture a sufficient range of the variations in defect geometry, position, and imaging conditions.
2. The images are correctly labeled by the subject matter experts.
3. The dataset is sufficient for a CNN to accurately learn the mapping function between inputs and outputs after applying data augmentation and transfer learning.

During training, it is assumed that there are no substantial improvements in model performance beyond the 50^{th} epoch.

CHAPTER 5. CONCLUSION

5.1 Contributions

In this thesis, a novel application of stacked ensembling to automated defect detection in aircraft surfaces is presented. The ensembles consist of three CNN base learners that are combined with a logistic regression meta-learner to make class predictions on input images. The CNNs are trained on images taken from a borescope inspection process for C130 propeller blades that contain defects of varying type, size, location, and illumination. The proposed framework demonstrates higher detection performance than the current-state-of-art and can be used to minimize missed detections while maintaining a low rate of false alarms for critical operations. The error reduction of the ensemble frameworks allows it to obtain a higher level of precision at near perfect or perfect recall than single model CNN systems. Error diversity in the base learners was established as a key factor in ensemble performance. Other algorithms such as SVMs, linear regressions, and ANNs can be used in place of the logistic regression as the meta-learner.

5.2 Limitations

A limitation of this ensemble framework are the computational requirements associated with training and running three CNNs in parallel. Despite downwards trends in GPU price-performance ratios and availability of cheap memory, the resources required can be cost prohibitive in some situations. The size of the dataset used in this study is another limitation. It is difficult to properly gauge an ensemble's generalization performance on a limited sample size and there may be inherent biases within the

framework that remain undetected. The defect rates of the defects found in the inspection process are relatively low, making it difficult to generate a large dataset; however, this is a common issue in the literature and techniques such as transfer learning and data augmentation are used to reduce the image requirements and increase dataset size. The ensembles in this study can be continually trained and evaluated on new images to better establish operational performance. Finally, the performance of the ensembles is highly dependent on the composition of the base learners. Diversity in model errors, structure, parameters, or training sets have been shown to factor heavily in an ensemble's generalization ability. Lack of sufficient diversity will result in a decrease in overall performance.

5.3 Future Work

This research can be further extended in four different ways: 1) The hyperparameters and confidence thresholds for the base learners can be further fine-tuned for each individual model. Hyperparameter selection was done at the network level for simplicity and confidence thresholds were left at 50%. With threshold adjustment, it can be hypothesized that a model with perfect precision, but low recall can be combined with a model with low precision but perfect recall to achieve perfect accuracy. This can be extended to multiple base learners to improve ensemble performance. 2) The training annotation can be extended to multi-class classifications where defects are labeled by type. 3) Use Generative Adversarial Networks [55] to generate artificial data to further train the networks. 4) Apply ensembles to object detection networks that localize defects within the surrounding image using bounding boxes. Current state-of-the-art in an application of Mask R-CNN to aircraft dents and achieves recall of 53.6% precision and 46.2% recall on

6 images [56]. Ensembles may be able to improve the performance of these networks by training on the bounding box predictions and combining multiple predictors.

APPENDIX A. ENSEMBLE CODE

This appendix contains the code used to train and evaluate the various components of the stacked ensembles. The code for CNN training and evaluation is adapted from the PyTorch tutorial for transfer learning found at:

https://pytorch.org/t utorials/beginner/transfer_learning_tutorial.html

A.1 CNN Training Code

```
from __future__ import print_function, division

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy

if __name__ == "__main__":
    feature_extract = False
    num_classes = 2
    data_transforms = {
        'train': transforms.Compose([
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ]),
        'val': transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ]),
    }

    data_dir = "C:/Users/Ivan/Datasets/Dataset3"
    image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                              data_transforms[x])
                      for x in ['train', 'val']}
    dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x],
batch_size=5,
```

```
                                                shuffle=True, num_workers=4)
                  for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                scheduler.step()
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    value, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # Statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]
```

```
                print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                    phase, epoch_loss, epoch_acc))

                # deep copy the model
                if phase == 'val' and epoch_acc > best_acc:
                    best_acc = epoch_acc
                    best_model_wts = copy.deepcopy(model.state_dict())

            print()

        time_elapsed = time.time() - since
        print('Training complete in {:.0f}m {:.0f}s'.format(
            time_elapsed // 60, time_elapsed % 60))
        print('Best val Acc: {:4f}'.format(best_acc))

        # load best model weights
        model.load_state_dict(best_model_wts)

        checkpoint = {'model': model_ft,
                      'state_dict': model.state_dict(),
                      'optimizer' : optimizer.state_dict()}

        torch.save(checkpoint, 'epoch.pth')
        return model

    def set_parameter_requires_grad(model, feature_extracting):
        if feature_extracting:
            for param in model.parameters():
                param.requires_grad = False

    def initialize_model(model_name, num_classes, feature_extract,
use_pretrained=True):
    # Initialize these variables which will be set in this if statement. Each
of these
    #   variables is model specific.
        model_ft = None
        input_size = 0

        if model_name == "resnet":
            """ Resnet18
            """
            model_ft = models.resnet18(pretrained=use_pretrained)
            set_parameter_requires_grad(model_ft, feature_extract)
            num_ftrs = model_ft.fc.in_features
            model_ft.fc = nn.Linear(num_ftrs, num_classes)
            input_size = 224

        elif model_name == "googlenet":
            """ GoogleNet
            """
            model_ft = models.googlenet(pretrained=use_pretrained)
            set_parameter_requires_grad(model_ft, feature_extract)
            num_ftrs = model_ft.fc.in_features
            model_ft.fc = nn.Linear(num_ftrs, num_classes)
            input_size = 224
```

```
        elif model_name == "vgg":
            """ VGG11_bn
            """
            model_ft = models.vgg11_bn(pretrained=use_pretrained)
            set_parameter_requires_grad(model_ft, feature_extract)
            num_ftrs = model_ft.classifier[6].in_features
            model_ft.classifier[6] = nn.Linear(num_ftrs,num_classes)
            input_size = 224

        else:
            print("Invalid model name, exiting...")
            exit()
        return model_ft, input_size

    # Initialize the model for this run
    model_ft, input_size = initialize_model("vgg",num_classes,
feature_extract, use_pretrained=True)
    model_ft = model_ft.to(device)

    params_to_update = model_ft.parameters()

    if feature_extract:
        params_to_update = []
        for name,param in model_ft.named_parameters():
            if param.requires_grad == True:
                params_to_update.append(param)

    ## Define the hyperparamters
    criterion = nn.CrossEntropyLoss()
    optimizer_conv = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
    exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7,
gamma=0.1)

    model_ft = train_model(model_ft, criterion, optimizer_conv,
                           exp_lr_scheduler, num_epochs=25)
```

A.2 CNN Evaluation Code

```
from __future__ import print_function, division

import torch
from torchvision import datasets, transforms
import os

if __name__ == "__main__":
    ## Load saved model
    def load_checkpoint(filepath):
        checkpoint = torch.load(filepath)
        model = checkpoint['model']
        model.load_state_dict(checkpoint['state_dict'])
        model.eval()
        return model
```

```
    model = load_checkpoint('ResNet18_Checkpoints/ResNet18_1.pth')
    data_dir = "C:/Users/Ivan/Datasets/Test"
    data_transforms = {
        'test': transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ]),
    }


    image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                              data_transforms[x])
                      for x in ['test']}
    dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x],
batch_size=40,
                                                  shuffle=False, num_workers=4)
                   for x in ['test']}
    dataset_sizes = {x: len(image_datasets[x]) for x in ['test']}
    class_names = image_datasets['test'].classes

    device = torch.device("cuda:0")

    def test_model(model, threshold):
        model.eval()
        preds = torch.tensor([0], device = device, dtype=torch.long)
        correct = 0

        with torch.no_grad():
            for i, (inputs, labels) in enumerate(dataloaders['test']):
                inputs = inputs.to(device)
                labels = labels.to(device)
                outputs = model(inputs)
                sm = torch.nn.Softmax(dim=1)
                prob = sm(outputs)
                conf = prob[0][0]
                if conf.item() <= threshold:
                    preds[0] = 1
                correct += (preds == labels).sum().item()


    test_model(model, 0.5)
```

A.3 Ensemble Training and Evaluation

```
# Import data analysis modules
import pandas as pd

# Load prediction data from Excel sheets
train1 = pd.read_excel('ResNet18.xlsx','Detailed Results')
train2 = pd.read_excel('GoogleNet.xlsx','Detailed Results')
train3 = pd.read_excel('VGG11bn.xlsx','Detailed Results')

# Select training instances
ResNet18 = train1[['ResNet18_1']]
```

```
GoogleNet = train2[['GoogleNet_2']]
VGG11bn = train3[['VGG11bn_3']]
X = pd.concat([ResNet18,GoogleNet,VGG11bn],axis=1)
y = train1['Truth (1 = Defect)']

# Define test and training sets
X_values = X
y_values = y

# Import module for fitting
from sklearn.linear_model import LogisticRegression
# Create instance (i.e. object) of LogisticRegression
logmodel = LogisticRegression()
# Fit the model using the training data
logmodel.fit(X_values, y_values)

# Test on data and output predictions + probabilities
proba = pd.DataFrame(logmodel.predict_proba(X_values))
pred = logmodel.predict(X_values)
```

APPENDIX B. STATISTICAL TEST RESULTS

Wilcoxon Ranked test results

H_0: Group 1 ≥ Group 2, H_1: Group 1 Group 2 α = 0.05, N = 10 for all Group 1 and 2						
Group 1	Group 2	N	Accuracy		F1 Score (%)	
			Z	*p*	Z	*p*
ResNet18	Ensemble A	10	-2.641	0.004	-2.611	0.004
ResNet34		10	-2.768	0.003	-2.754	0.003
ResNet50		10	-2.796	0.002	-2.756	0.003
GoogLeNet		10	-2.768	0.003	-2.754	0.003
VGG11_bn		10	-2.771	0.003	-2.756	0.003
ResNet18	Ensemble B	10	-2.663	0.004	-2.611	0.004
ResNet34		10	-2.773	0.003	-2.756	0.003
ResNet50		10	-2.819	0.002	-2.756	0.003
GoogLeNet		10	-2.757	0.003	-2.754	0.003
VGG11_bn		10	-2.773	0.003	-2.754	0.003
ResNet18	Ensemble C	10	-2.401	0.008	-2.347	0.009
ResNet34		10	-2.627	0.004	-2.754	0.003
ResNet50		10	-2.627	0.004	-2.754	0.003
GoogLeNet		10	-2.762	0.002	-2.752	0.003
VGG11_bn		10	-2.763	0.002	-2.756	0.003

REFERENCES

[1] Federal Aviation Administration, “Visual Inspection for Aircraft.” 14-Aug-1997.

[2] W. Chen and S. Huang, “Human Reliability Analysis for Visual Inspection in Aviation Maintenance by a Bayesian Network Approach,” *Transportation Research Record*, vol. 2449, no. 1, pp. 105–113, Jan. 2014.

[3] Zachary Hansen, “Military plane crash that killed 16 blamed on faulty maintenance at Georgia air base,” *The Atlanta Journal Constitution*, 05-Dec-2018. [Online], Available: https://www.ajc.com/news/breaking-news/military-plane-crash-that-killed-blamed-faulty-maintenance-georgia-air-base/vmjvKVqj64WGK4DvzFankL/. [Accessed March 10, 2020].

[4] X. Xie, “A Review of Recent Advances in Surface Defect Detection using Texture analysis Techniques,” *ELCVIA*, vol. 7, no. 3, p. 1, Jun. 2008.

[5] Z. Dworakowski, K. Dragan, and T. Stepinski, “Artificial neural network ensembles for fatigue damage detection in aircraft,” *Journal of Intelligent Material Systems and Structures*, vol. 28, no. 7, pp. 851–861, Apr. 2017.

[6] C. G. Drury and J. Watson, “Good Practices in Visual Inspection,” *Aircraft Maintenance Technology*, pp. 74–76.

[7] “Innovation takes aircraft visual inspections to new heights,” *Airbus*, 10-Apr-2018. [Online].Available: https://www.airbus.com/newsroom/news/en/2018/04/innovation-takes-aircraft-visual-inspections-to-new-heights.html. [Accessed: 13-Mar-2020].

[8] J. E. See, “Visual Inspection: A Review of the Literature,” , 2012.

[9] E. C. Poulton, “The effect of fatigue upon inspection work,” *Applied Ergonomics*, vol. 4, no. 2, pp. 73–83, Jun. 1973.

[10] Keith H. Nuechterlein, Raja Parasuraman, and Qiyuan Jiang, “Visual Sustained Attention: Image Degradation Produces Rapid Sensitivity Decrement over Time,” *Science*, vol. 220, no. 4594, pp. 327–329, Apr. 1983.

[11] Anand K. Gramopadhye, Colin G. Drury, and Joseph Sharit, “Feedback strategies for visual search in airframe structural inspection,” *International Journal of Industrial Ergonomics*, vol. 19, no. 5, pp. 333–334, Jan. 1996.

[12] S. Lee, L.-M. Chang, and M. Skibniewski, “Automated recognition of surface defects using digital color image processing,” *Automation in Construction*, vol. 15, no. 4, pp. 540–549, Jul. 2006, doi: 10.1016/j.autcon.2005.08.001.

[13] H.-F. Ng, “Automatic thresholding for defect detection,” *Pattern Recognition Letters*, vol. 27, no. 14, pp. 1644–1649, Oct. 2006, doi: 10.1016/j.patrec.2006.03.009.

[14] Robert M. Haralick, K. Shanmugam, and I. Dinstein, “Textural Features for Image Classification,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 3, no. 6, pp. 610–621, Nov. 1973.

[15] P. Caleb and M. Steuer, “Classification of Surface Defects on Hot Rolled Steel Using Adaptive Learning Methods,” in *Proc. IEEE Fourth Int. Conf. on Knowledge-Based Intelligent Engineering Systems & Allied Technologies*, 2000, pp. 103–108.

[16] B. Tang, J. Kong, X. Wang, and L. Chen, “Surface Inspection System of Steel Strip Based on Machine Vision,” in *2009 First International Workshop on Database Technology and Applications*, 2009, pp. 359–362, doi: 10.1109/DBTA.2009.133.

[17] F. Bonnin-Pascual and A. Ortiz, “Combination of weak classifiers for metallic corrosion detection and guided crack location,” in *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)*, Bilbao, 2010, pp. 1–4, doi: 10.1109/ETFA.2010.5641267.

[18] Yafei Wang and Guangxu Cheng, “Application of gradient-based Hough transform to the detection of corrosion pits in optical images,” *Applied Surface Science*, vol. 366, pp. 9–18, 2016, doi: 10.1016/j.apsusc.2015.12.207.

[19] Y.-J. Jeon, J. P. Yun, D. Choi, and S. W. Kim, “Defect detection algorithm for corner cracks in steel billet using discrete wavelet transform,” in *2009 ICCAS-SICE*, 2009, pp. 2769–2773.

[20] J. H. Guo, X. D. Meng, and M. D. Xiong, “Study on Defection Segmentation for Steel Surface Image Based on Image Edge Detection and Fisher Discriminant,” *J. Phys.: Conf. Ser.*, vol. 48, pp. 364–368, Oct. 2006, doi: 10.1088/1742-6596/48/1/068.

[21] M. Mumtaz, A. B. Mansoor, and H. Masood, “A New Approach to Aircraft Surface Inspection Based on Directional Energies of Texture,” in *2010 20th International Conference on Pattern Recognition*, Istanbul, Turkey, 2010, pp. 4404–4407, doi: 10.1109/ICPR.2010.1070.

[22] Z. Liu, D. S. Forsyth, A. Marincak, and P. Vesley, “Automated rivet detection in the EOL image for aircraft lap joints inspection,” *NDT & E International*, vol. 39, no. 6, pp. 441–448, Sep. 2006, doi: 10.1016/j.ndteint.2006.01.002.

[23] P. Gunatilake, M. Siegel, A. G. Jordan, and G. W. Podnar, “Image understanding algorithms for remote visual inspection of aircraft surfaces,” presented at the Electronic Imaging ’97, San Jose, CA, 1997, pp. 2–13, doi: 10.1117/12.271231.

[24] M. R. Jahanshahi and S. F. Masri, “Effect of Color Space, Color Channels, and Sub-Image Block Size on the Performance of Wavelet-Based Texture Analysis Algorithms: An Application to Corrosion Detection on Steel Structures,” in *Computing in Civil Engineering*, Los Angeles, California, 2013, pp. 685–692, doi: 10.1061/9780784413029.086.

[25] C. Mandriota, M. Nitti, N. Ancona, E. Stella, and A. Distante, “Filter-based feature selection for rail defect detection,” *Machine Vision and Applications*, vol. 15, no. 4, pp. 179–185, Oct. 2004, doi: 10.1007/s00138-004-0148-3

[26] C. Unsalan and A. Erçil, “Automated Inspection of Steel Structures,” *Recent Advances in Mechatronics*, pp. 468–480, 1999.

[27] L. A. O. Martins, F. L. C. Pádua, and P. E. M. Almeida, “Automatic detection of surface defects on rolled steel using Computer Vision and Artificial Neural Networks,” in *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, 2010, pp. 1081–1086, doi: 10.1109/IECON.2010.5675519.

[28] Hongbin Jia, Y. L. Murphey, Jinajun Shi, and Tzyy-Shuh Chang, “An intelligent real-time vision system for surface defect detection,” in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, Cambridge, UK, 2004, pp. 239-242 Vol.3, doi: 10.1109/ICPR.2004.1334512.

[29] N.-D. Hoang and V.-D. Tran, “Image Processing-Based Detection of Pipe Corrosion Using Texture Analysis and Metaheuristic-Optimized Machine Learning Approach,” *Computational Intelligence and Neuroscience*, 2019. [Online]. Available: https://www.hindawi.com/journals/cin/2019/8097213/. [Accessed: 17-Mar-2020].

[30] Xing Zhiwei, Chen Muhua, and Gao Qingji, “The structure and defects recognition algorithm of an aircraft surface defects inspection robot,” in *2009 International Conference on Information and Automation*, Zhuhai/Macau, China, 2009, pp. 740–745, doi: 10.1109/ICINFA.2009.5205019.

[31] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[32] D. J. Atha and M. R. Jahanshahi, “Evaluation of deep learning approaches based on convolutional neural networks for corrosion detection,” *Structural Health Monitoring*, vol. 17, no. 5, pp. 1110–1128, Sep. 2018, doi: 10.1177/1475921717737051.

[33] T. Malekzadeh, M. Abdollahzadeh, H. Nejati, and N.-M. Cheung, “Aircraft Fuselage Defect Detection Using Deep Neural Networks,” p. 5, 2017.

[34] Z. Shen, X. Wan, F. Ye, X. Guan, and S. Liu, “Deep Learning based Framework for Automatic Damage Detection in Aircraft Engine Borescope Inspection,” in *2019 International Conference on Computing, Networking and Communications (ICNC)*, 2019, pp. 1005–1010, doi: 10.1109/ICCNC.2019.8685593.

[35] B. Ramalingam *et al.*, “Visual Inspection of the Aircraft Surface Using a Teleoperated Reconfigurable Climbing Robot and Enhanced Deep Learning Technique,” *International Journal of Aerospace Engineering*, vol. 2019, pp. 1–14, Sep. 2019, doi: 10.1155/2019/5137139.

[36] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,” *Medium*, 17-Dec-2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed: 20-Mar-2020].

[37] “CS231n Convolutional Neural Networks for Visual Recognition.” [Online]. Available: http://cs231n.github.io/neural-networks-1/. [Accessed: 20-Mar-2020].

[38] “Intro to optimization in deep learning: Gradient Descent,” *Paperspace Blog*, 02-Jun-2018. [Online]. Available: https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/. [Accessed: 20-Mar-2020].

[39] S. Sun, Z. Cao, H. Zhu, and J. Zhao, “A Survey of Optimization Methods from a Machine Learning Perspective,” *arXiv:1906.06821 [cs, math, stat]*, Oct. 2019.

[40] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255, doi: 10.1109/CVPR.2009.5206848.

[41] Y. LeCun *et al.*, “Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989, doi: 10.1162/neco.1989.1.4.541.

[42] R. Geirhos, D. H. J. Janssen, H. H. Schütt, J. Rauber, M. Bethge, and F. A. Wichmann, “Comparing deep neural networks against humans: object recognition when the signal gets weaker,” *arXiv:1706.06969 [cs, q-bio, stat]*, Dec. 2018.

[43] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv:1409.1556 [cs]*, Apr. 2015.

[44] C. Szegedy *et al.*, “Going Deeper with Convolutions,” *arXiv:1409.4842 [cs]*, Sep. 2014.

[45] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv:1512.03385 [cs]*, Dec. 2015.

[46] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” *arXiv:1502.03167 [cs]*, Mar. 2015.

[47] L. K. Hansen and P. Salamon, “Neural network ensembles,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, no. 10, pp. 993–1001, Oct. 1990, doi: 10.1109/34.58871.

[48] L. Liu *et al.*, “Deep Neural Network Ensembles against Deception: Ensemble Diversity, Accuracy and Robustness,” p. 9, 2019.

[49] L. Breiman, “Bagging predictors,” *Machine Learning*, vol. 24, no. 2, pp. 123–140, Aug. 1996, doi: 10.1007/BF00058655.

[50] Y. Freund and R. E. Schapire, “Experiments with a New Boosting Algorithm,” 1996.

[51] D. Wolpert, “Stacked Generalization,” *Neural Networks*, vol. 5, no. 2, pp. 241–259, Jul. 1991, doi: 10.1016/S0893-6080(05)80023-1.

[52] Y. He, K. Song, Q. Meng, and Y. Yan, “An End-to-end Steel Surface Defect Detection Approach via Fusing Multiple Hierarchical Features,” *IEEE Transactions on Instrumentation and Measurement*, pp. 1–1, 2019, doi: 10.1109/TIM.2019.2915404.

[53] “PyTorch.” https://www.pytorch.org (accessed Mar. 24, 2020).

[54] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol 12 , 2011.

[55] A. Radford, L. Metz, and S. Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” arXiv:1511.06434 [cs], Jan. 2016.

[56] S. Bouarfa, A. Doğru, R. Arizar, R. Aydoğan, and J. Serafico, “Towards Automated Aircraft Maintenance Inspection. A use case of detecting aircraft dents using Mask R-CNN,” in AIAA Scitech 2020 Forum, Orlando, FL, 2020, doi: 10.2514/6.2020-0389.